AD-A261 287

DTIC
ELECTE
FEB 12 1993
A S E D

Facsimile
Report
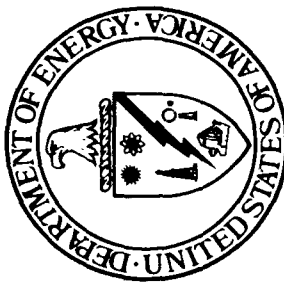
93   2   10   101

93-02585

MASTER

SAND--86-1048C

DE86 012289

FINAL REPORT

Of The

PDES LOGICAL LAYER INITIATION TASK

Submitted To

Kalman Brauner, PDES Chairman

The Boeing Commercial Airplane Company

April 28, 1986

Accesion For	
NTIS CRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and / or Special
A-1	

DTIC QUALITY INSPECTED 3

Submitted By

J. C. Kelly, Chairman, PDES Logical
Layer Initiation Task Group

Sandia National Laboratories

DTIC QUALITY INSPECTED 3

DISCLAIMER

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

Table of Contents

FOREWORD

This report deals with the activities, findings, conclusions, and recommendations of the Logical Layer Initiation Task Group, one of two tasks groups comprising the PDES Initiation effort. The PDES initiation effort is a proof of concept effort with a subordinate purpose of producing content potentially suitable for PDES Version 1.0. The primary goal of the Logical Layer Initiation Task Group was to formulate and test a methodology for developing the PDES specification along the lines advocated in the Second PDES Report. The Task Group was also charged with making a final report detailing its "lessons learned" and its recommendations. This report addresses that charge.

The Logical Layer Initiation Task Group was chaired by J. C. Kelly of Sandia National Laboratories. John Zimmerman of the Bendix Kansas City Division of Allied Corporation, and Doug Schenck of McDonnell Douglas Aerospace Information Services Company formed the technical core within the Logical Layer itself. In terms of the declared content deliverables of the project, Zimmerman developed the "qualified" NIAM information models of the application areas and the global NIAM information model of the Logical Layer. Schenck developed the Data Specification Language and the expression of the global models in that language.

Many other people contributed to the goals of the Task Group work by participating in the development of application area information models which were then used to exercise the methodology. The primary contributors to the development of these models and their sponsoring companies are named in Figures 1-3 and 1-5 in Section 1.4. Appendix A contains valuable retrospective comments on the experience of the initiation effort by the modelers of three application areas and one resource area.

Of all the contributors to the Task Group work, John Zimmerman deserves special mention. He has worked longer and harder than anyone else associated with the project, and has had the greatest technical impact. He defined the methodology and provided the information modeling expertise to carry it out. He liased with the various application layer groups, assimilated their models, and performed the crucial integration task yielding the global models supporting all the applications. He is the primary technical contributor to this report. In short, he has been the one with whom the technical buck has stopped most often during this project.

The Logical Layer initiation effort formally began in January, 1985 with a meeting at Boeing. The effort will end with the publication of this report. The existence of the Task Group was publicized, and a preliminary statement of purpose was given,

in the General Assembly at the quarterly meeting of the IGES Committee in Pomona in February, 1985. It was announced that membership in the Task Group was open to any individual who could commit to attending a rather intensive schedule of working meetings and to significant assignments to be completed between meetings. The sense of urgency surrounding the project, and, for that matter, surrounding the entire initiation effort, derives from a commitment to develop a single worldwide standard for product data exchange within the ISO TC184/SC4 community as soon as possible. (TC184 is Industrial Automation Systems; SC4 is External Representation Of Product Model Data; STEP, the eventual worldwide standard, is Standard For The Transfer And Exchange Of Product Model Data.) The PDES Project has been designated as the mechanism by which the United States will contribute to the STEP effort.

Since January, 1985, scheduled Logical Layer Initiation meetings have been held as follows:

> Seattle - Working Meeting - January, 1985
> Pomona - Quarterly IGES Meeting - February, 1985
> Cincinnati - Working Meeting - February, 1985
> Kansas City - Working Meeting - March, 1985
> Atlanta - Quarterly IGES Meeting - April, 1985
> St. Louis - Working Meeting - May, 1985
> Albuquerque - Working Meeting - June, 1985
> Madison - Quarterly IGES Meeting - July, 1985
> Los Angeles - Working Meeting - September, 1985
> Knoxville - Quarterly IGES Meeting - October, 1985
> Kansas City - Working Meeting - November, 1985
> San Diego - Quarterly IGES Meeting - January, 1986
> Albuquerque - Working Meeting - February, 1986

In addition, other, more informal, meetings were held. These include, for example, liason meetings between Zimmerman and people from the application areas, and meetings concerned with this report. And, it would be difficult to estimate the number of lengthy telephone conversations held in conjunction with the project, but the number is assuredly quite high.

Meetings associated with the regular quarterly IGES meetings amounted to organized progress reports to a group that has since seen some stabilization and in fact now forms the Logical Layer Committee. Several people with considerable experience in information modeling and logical database design have been attracted to this group. Detailed minutes were published following each quarterly IGES meeting, and these were supplied to the ISO Working Group TC184/SC4/WG1 as well.

The work in the presentation area of Richard C. Winfrey of Digital Equipment Corporation was used as a basis for establishing communication with the ANSI X3H3 Committee

responsible for the development of the PHIGS graphics standard. It is believed that this type of coordination between the IGES/PDES Committee and other standards development groups for the purpose of minimizing duplication will be commonplace in the future.

A talk on PDES in general, and the initiation effort in particular, was developed and presented at the Federal Computer Conference in Washington in September, 1985, at the PDDI end-of-contract briefing in St. Louis in September, 1985, and at the SME CIMTECH conference in Boston in March, 1986. A paper was developed along the same lines and was published in Manufacturing Productivity FRONTIERS, a publication of the IIT Research Institute, and in the CIMTECH conference proceedings. The paper is included at the end of this report as Appendix E.

This report has been written with the members of the IGES Committee in mind. The extensive set of Appendices indicates that we feel the report has an archival function to perform. An effort has been made to keep "gory details" in the appendices.

I believe that the Logical Layer Initiation Task Group has performed as needed in a proof of concept effort. I say this in spite of the fact that, because of time and manpower constraints, we have had to cut short our efforts in some areas, and have had to leave open some issues that have been raised. However, we did define a project along the lines of the Second PDES Report and carry it out. We did define a methodology, we have refined it in light of what we consider to be authentic experience, and we are able to recommend it. We did gain some experience and we do have some lessons learned to relate and some recommendations to give. And, we did cause some of the standing IGES committees to mobilize and gain some experience with information modeling. We believe these are all positive contributions toward the development of PDES Version 1.0.

Dixie Chavez of Sandia National Laboratories and Will Williams of Bendix, Kansas City Division deserve special thanks for their help in preparing this report. Our wives and families deserve our thanks for helping us live through the initiation experience, and we congratulate them for living through it themselves.

I speak for the entire Task Group when I say we are proud to present the results of our work to the IGES/PDES Committee. We look forward to continuing participation in the development of PDES, and we wish Doug Schenck great success in his new role of Logical Layer Chairman.

> J. C. Kelly, Chairman
> PDES Logical Layer Initiation Task Group

1. Introduction

1.1 The Mission Of The PDES Initiation Effort

The PDES initiation effort is a limited term proof of concept effort. The intent is to gain initial experience in developing an exchange specification for product data in accordance with what was proposed in the Second PDES Report (See Reference 1). The emphasis is on establishing a development methodology and on reporting back lessons learned and recommendations. A lesser emphasis is on developing actual content for longer term PDES work; however, it is felt that the initiation effort has produced work that is legitimate and valuable from a content point of view.

The general spirit regarding the output of the initiation effort is that what is good will be retained for longer term PDES work. All results from the initiation effort are subject to scrutiny and modification prior to possible acceptance.

The initiation effort was administered by two task groups, the Logical Layer Initiation Task Group and the Physical File Structure and Formal Language Committee. This report concentrates on the activities, findings, conclusions, and recommendations of the Logical Layer Initiation Task Group.

Beginning the overall PDES effort with a proof of concept effort is justified. In addition to major differences between the scopes of the IGES specification and the envisioned PDES specification, and in the intelligibility and sophistication of the data to be exchanged, there are also major changes in the process by which the PDES specification is to be developed. Distinguishing features in this process are given in the next subsection. In particular, development of the PDES specification will require settling on a methodology by which the various organizational components can work together. The major emphasis of the Logical Layer Initiation Task is to gain experience and make recommendations concerning a methodology for the development process for the PDES specification.

1.2 The Consistency Between The initiation effort And The Second PDES Report

The process for the development of the PDES specification that was followed in the initiation effort is consistent with the requirements given in the Second PDES Report, a report commissioned within the IGES Edit Committee and issued by the

PDES Chairman in late 1984. This report was presented to the IGES Steering Committee in early 1985.

The distinguishing features of the process according to that report are: use of reference models, use of formal languages, a minimally redundant entity set, and a three layer architecture.

A reference model is a mechanism for describing information. The development process outlined in the Second PDES Report placed great emphasis on the use of reference models. Information models such as those produced by the IDEF-1 and the NIAM information modeling techniques are examples of reference models. These modeling techniques are described in primers in appendix D1 and D2 respectively.

The three layer architecture involves an application layer, a logical layer, and a physical layer. The following notions are adequate for the introductory purpose here: The application layer consists of reference models constructed for various application areas using information modeling techniques. These models describe product data for the various areas from the respective user points of view. The logical layer consists of a reference model describing a minimally redundant set of generic entities and structures that play a supporting role in describing product data. This logical layer reference model will be referred to in this report as the "Logical Layer Model." The specific manner in which these entities carry out this role is also of importance. The physical layer consists of a reference model defining the actual physical format structure by which product data will be exchanged.

Thus the main pieces, and the flow, in the PDES specification development process are:

(i)   the user - the process starts with precise, user developed descriptions of the product data to be exchanged

(ii) the logical method - the process continues with the integration of the piecemeal descriptions of product data into a logical whole, and with the identification and specification of the manner in which the information is to be carried within a bounded set of logical structures.

(iii) the format - the process terminates with the imbedding of the descriptions of the product data into a physical format structure.

The initiation effort has incorporated the distinguishing features of PDES. It has involved these three pieces, and has exercised the flow between them. Reverse-flow loops have been used for assuring quality. Information models have been produced and used in the context of the three layer architecture. In

2

particular, the activities of the Logical Layer Initiation Task
Group have involved the generation of application layer
information, and have also involved specific deliverables to the
Physical File Committee.

1.3 The Three Layers - Application Logical, And Physical

It is appropriate to summarize the descriptions of the three
layers as given in the Second PDES Report. See Figure 1-1,
slightly modified from the Second PDES Report, for a depiction of
the layers.

The top layer is the user or application layer. This is the
layer at which the ultimate user lives and thinks. He formulates
his data requirements in his own terms stating concisely what he
needs. He draws from his own experience and from the established
terms, conventions, techniques, and methodologies of his
discipline. The different application groups such as Electrical
Products or Finite Element Modeling define the information
entities relevant to their application, and construct reference
models for them. This layer of the specification contains as
many different applications and entities within those
applications as there is apparent need for.

The second layer is the logical or conceptual layer. This
is where the data content for the set of generic entities and
structures is defined. This set should be a normalized,
minimumly redundant set that supports (is a basic resource for)
the information defined by the applications. At this layer,
logical commonality will be sought across all applications.
Complex things, events, and phenomena in the application layer
will be constructed from less complex things, events, and
phenomena in the logical layer whenever possible. Similarities
in the information requirements of different applications will be
integrated into single conceptual entities. The integration of
different application requirements will control the definition of
redundant entities and will help to ensure a consistent, coherent
entity set. The entity definitions at this layer will be in
logical form in a reference model.

The bottom layer is the physical or internal layer. This
layer contains one or more actual file format definitions. It
consists of the description of the sections, records, fields,
sequencing, and associated formats for the exchange file. A
formal definition language reference model will be used to reduce
ambiguity.

1.4 The Logical Layer Initiation Charter

A mission for the Logical Layer Initiation Task Group
consistent with the Second PDES Report was developed in the form

APPLICATION LAYER

LOGICAL LAYER

PHYSICAL LAYER

APPLICATION N

FINITE ELEMENT MODELING

ELECTRICAL PRODUCTS

MECHANICAL PRODUCTS

GENERIC ENTITIES

FILE FORMAT AND DATA STRUCTURE
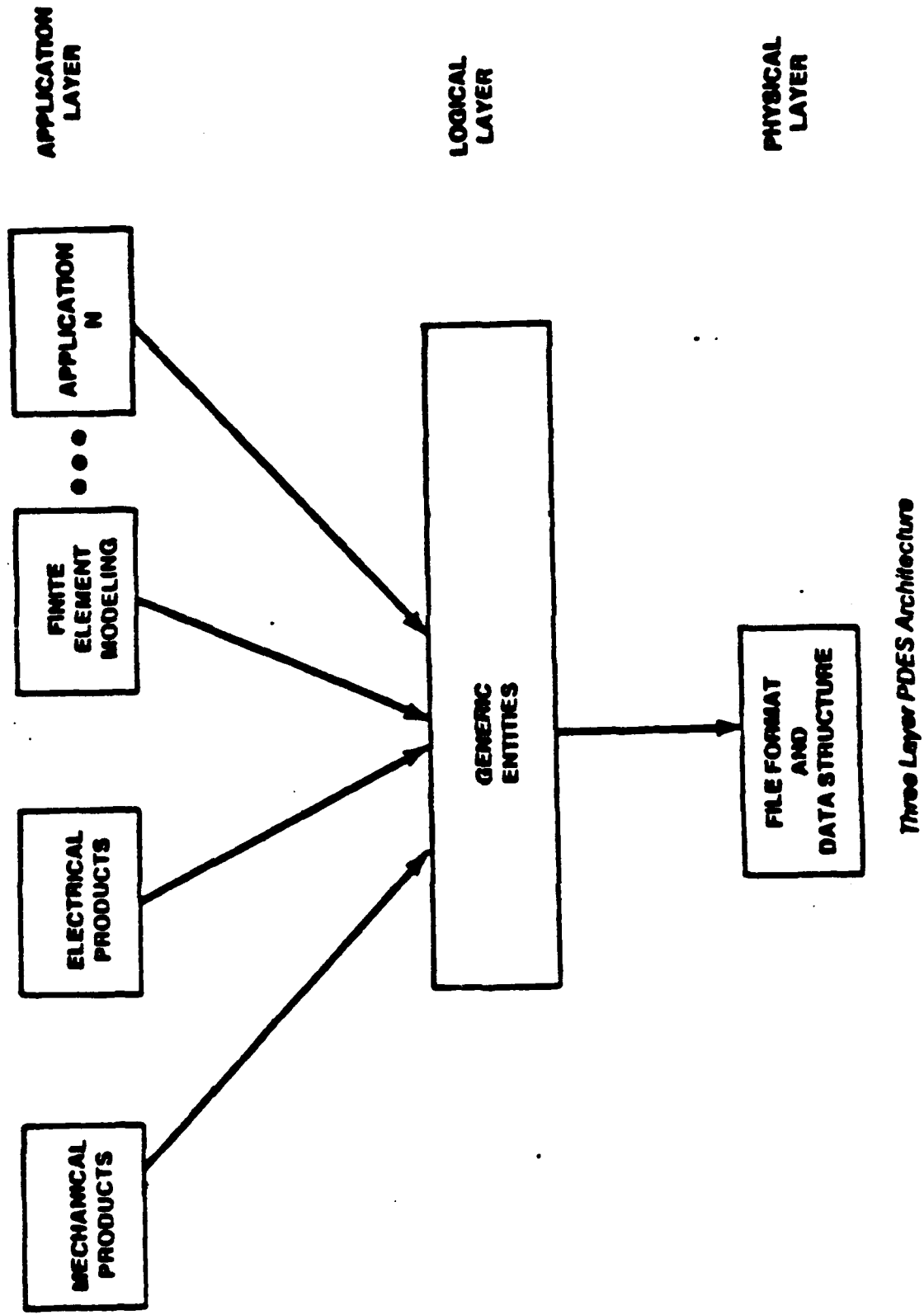
Three Layer PDES Architecture

Figure 1-1

of a charter. The charter was then followed to the extent possible for the remainder of the effort. The charter is included in this report as Appendix F.

The two main tasks adopted by the Task Group were:

Task 1: Develop a Logical Layer model that supports one specific application area model, and then communicate the three part schema consisting of the two models and their interrelationships to the Physical File Structure and Formal Languages Committee.

Task 2: Further develop the Logical Layer model to support additional application area models, and communicate the three part schema associated with each of these application area models to the Physical File Structure and Formal Languages Committee.

Thus, the Logical Layer model would serve as a resource to all the application area models. Each three part schema would be expressed first as an information model using the Nijssen Information Analysis Method (NIAM), and then communicated to the Physical File Committee in the form of the Data Specification Language (DSL). DSL is described in a primer in appendix D3.

The intent of the first task was to cause to be used all layers of the three layer architecture. The intent of the second task was to illustrate that the Logical Layer model could be expanded as the need arose. This was taken as a characteristic of the future PDES working environment.

The contents of the two tasks as actually carried out are presented in Figures 1-2 and 1-4. The primary contributors to the two tasks are presented in Figures 1-3 and 1-5.

For Task 1, a schema was developed supporting the mechanical products application area of flat plates with holes. A wireframe geometry model and a presentation (graphics) model were developed and served as the principal parts of the initial Logical Layer model. The flat plate application area itself served primarily as a vehicle. The primary intent was to choose something simple that would enable the work to get started.

For Task 2, three additional application area reference models were developed. The three application models were: Finite Element Modeling, Electrical Schematics, and Tolerancing. To support these additional application models, the initial

4

OVERVIEW OF TASK 1

LOGICAL LAYER

FORMAL LANGUAGE REFERENCE MODEL OF WIRE FRAME GEO.	IDEF 1 REFERENCE MODEL OF FLAT PLATE MECH. PARTS	FORMAL LANGUAGE REFERENCE MODEL OF PRESENTATION
QUALIFIED NIAM REFERENCE MODEL OF WIRE FRAME GEO.	QUALIFIED NIAM REFERENCE MODEL OF FLAT PLATE PARTS	QUALIFIED NIAM REFERENCE MODEL OF PRESENTATION

INTEGRATED NIAM REFERENCE MODEL - TASK 1 CONCEPTUAL SCHEMA -

DSL DESCRIPTION OF TASK 1 CONCEPTUAL SCHEMA

PHYSICAL DATA STRUCTURE

Fig. 1-2

TASK 1 PRIMARY CONTRIBUTORS

WIREFRAME GEOMETRY EDWARD CLAPP, IBM CURVES & SURFACES	FLAT PLATE PARTS SPENCER DE PAUW, CATERPILLAR	PRESENTATION R.C. WINFREY, DEC

```
QUALIFIED MODEL
PAUL THOMPSON, CDC
JOHN ZIMMERMAN
```
→
```
QUALIFIED MODEL
NOEL CHRISTENSEN,
JOHN ZIMMERMAN A/B
```
→
```
QUALIFIED MODEL
JOHN ZIMMERMAN
```

INTEGRATED NIAM REFERENCE MODEL
FOR CONCEPTUAL SCHEMA
JOHN ZIMMERMAN, BENDIX, KC DIV.

→ DSL DEVELOPMENT, AND DESCRIPTION OF
CONCEPTUAL SCHEMA IN DSL
DOUG SCHENCK, MC DONNELL DOUGLAS

→ DEVELOPMENT OF PHYSICAL DATA STRUCTURE
PHYSICAL FILE STRUCTURE AND FORMAL LANGUAGE COMMITTEE
W.B. GRUTTKE, CHAIRMAN, MC DONNELL DOUGLAS

Figure 1-3

OVERVIEW OF TASK 2

```
┌──────────────────────┐  ┌──────────────────────┐  ┌──────────────────────┐  ┌──────────────────────┐
│ IDEF 1 REFERENCE     │  │ IDEF 1/FORMAL        │  │ IDEF 1 REFERENCE     │  │ FORMAL LANGUAGE REF. │
│ MODEL OF FEM         │  │ LANGUAGE             │  │ MODEL OF ELECTRICAL  │  │ MODEL OF TOPOLOGY    │
│ APPLICATION          │  │ REFERENCE MODEL OF   │  │ APPLICATION          │  │                      │
│                      │  │ TOLERANCE APPLICATION│  │                      │  │                      │
└──────────────────────┘  └──────────────────────┘  └──────────────────────┘  └──────────────────────┘
```

LOGICAL LAYER

QUALIFIED NIAM REFERENCE MODEL OF FEM

QUALIFIED NIAM REFERENCE MODEL OF TOLERANCE

QUALIFIED NIAM REFERENCE MODEL OF ELECTRICAL

QUALIFIED NIAM REFERENCE MODEL OF TOPOLOGY

INTEGRATED NIAM REFERENCE MODEL AUGMENTING TASK 1 CONCEPTUAL SCHEMA
- THIS IS TASK 2 CONCEPTUAL SCHEMA -

DSL DESCRIPTION
OF
TASK 2 CONCEPTUAL SCHEMA

PHYSICAL DATA STRUCTURE

Figure 1-4

TASK 2 PRIMARY CONTRIBUTORS

FEM
R. IVEY, WESTINGHOUSE
FEM COMMITTEE

QUALIFIED MODEL
W. FREEMAN,
J. ZIMMERMAN A/B

TOLERANCE
R. COLSHER, IDA
W.B. BURKETT, MDC

QUALIFIED MODEL
W.B. BURKETT, MDC
J. ZIMMERMAN

ELECTRICAL
C.PARKS,GEN. DYNAMICS
ELECTRICAL COMMITTEE

QUALIFIED MODEL
C. PARKS, G D
J. ZIMMERMAN

TOPOLOGY
IGES ADV. GEOMETRY
COMMITTEE

QUALIFIED MODEL
J. ZIMMERMAN

INTEGRATED NIAM REFERENCE MODEL
FOR CONCEPTUAL SCHEMA
JOHN ZIMMERMAN, BENDIX, KC DIV.

DSL DEVELOPMENT, AND DESCRIPTION OF
CONCEPTUAL SCHEMA IN DSL
DOUG SCHENCK, MC DONNELL DOUGLAS

DEVELOPMENT OF PHYSICAL DATA STRUCTURE
PHYSICAL FILE STRUCTURE AND FORMAL LANGUAGE COMMITTEE
W.B. GRUTTKE, CHAIRMAN, MC DONNELL DOUGLAS

Figure 1-5

Logical Layer model was further developed to include topology and geometry-topology associativities.

1.5   Some Perspectives On Data Exchange In The PDES Environment

The work within the initiation effort has been directed toward a methodology for the development of the PDES specification according to the general guidelines of the Second PDES Report. That is, no work has been done that involves actual data exchange. Nonetheless, since data exchange is the ultimate task wi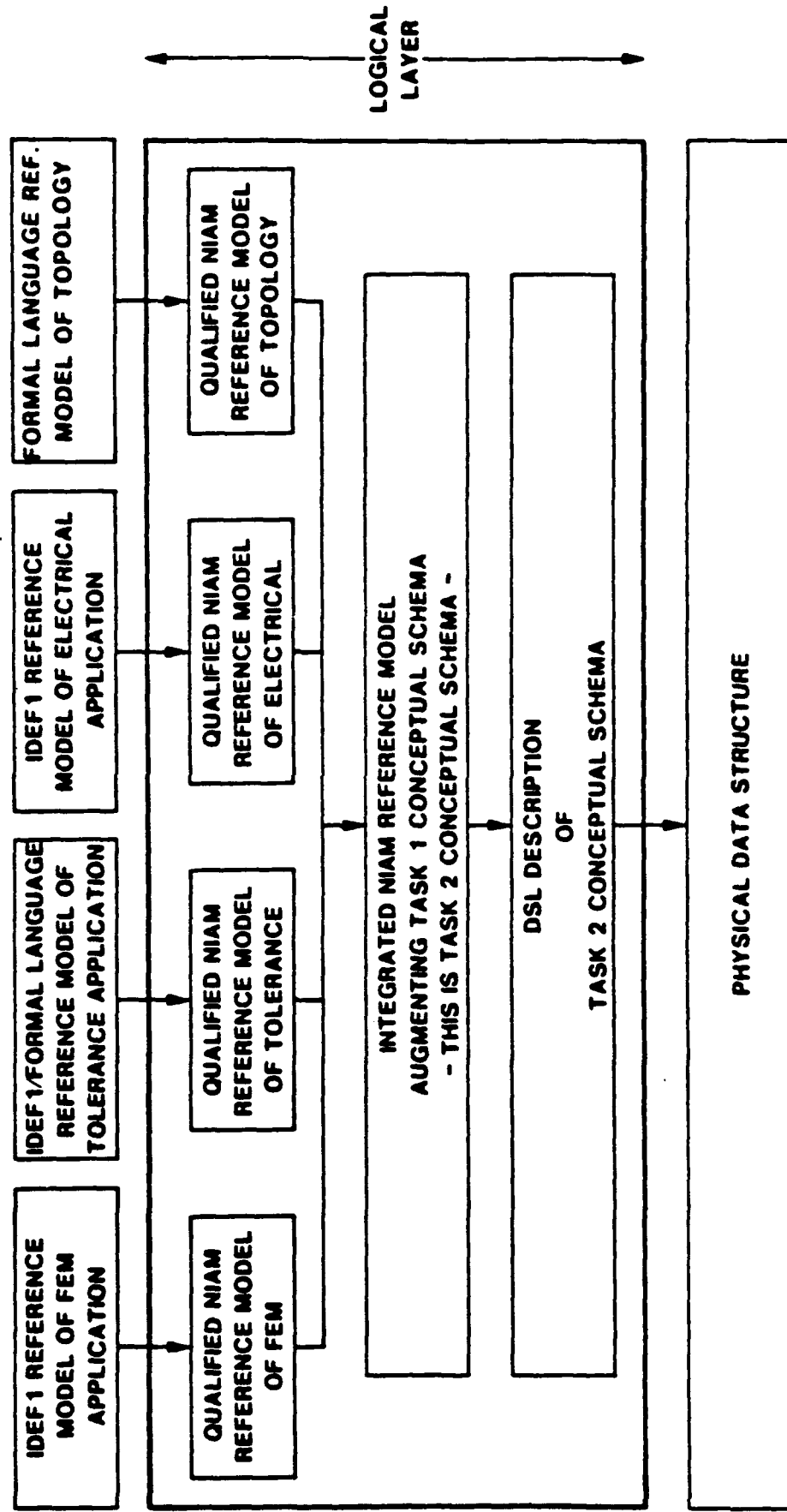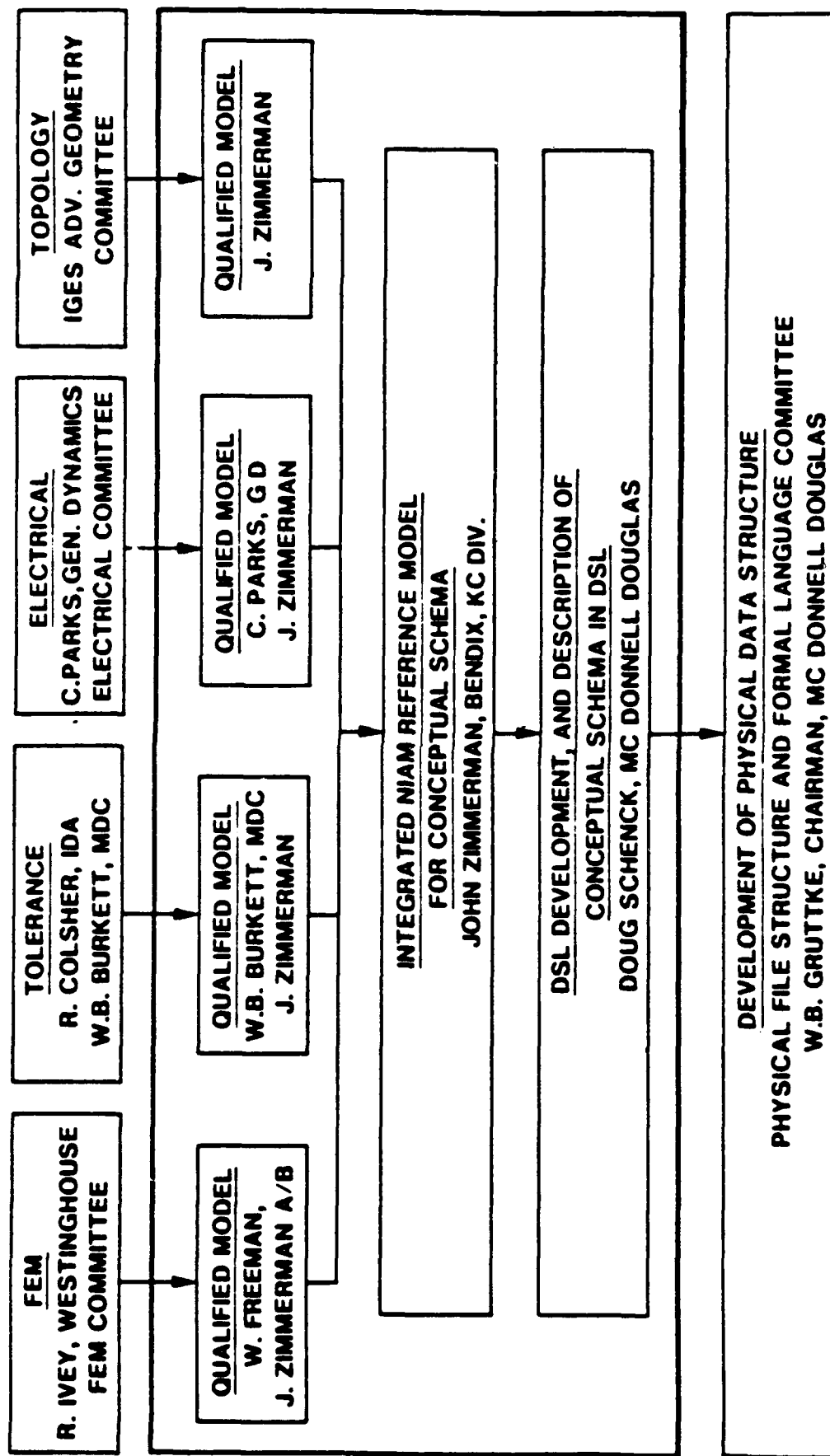th which the PDES effort must concern itself, we feel it is appropriate to discuss this topic and to pass along our current understandings, caveats, and recommendations. In addition, discussion of this topic enables us to place our initiation work in some global perspective. Unfortunately, some of the material is quite technically oriented for an introductory section. However, we feel it is also extremely basic, and that this is the logical location for it, and so have resisted all intermittent urges to locate it elsewhere.

1.5.1   Mental Models, Discipline-Specific Entities, Generic Entities, And Data Exchange Units

Data exchange is predicated upon the common awareness between a sender and a receiver of a "mental model". That is, the sender originates his data in terms of some model which makes the data meaningful to him. In order to recover the correct meaning of the data in an exchange, the receiver must share the same mental model as the sender.

For example, in the use of IGES, drawing data could be (and is) exchanged based on various mental models, such as the ANSI Y14.5 standard for dimensioning and tolerancing, MIL-STD 100 for drawings, various company standards, etc. It so happens that each of these mental models has been developed externally to IGES.

With PDES, product data will also be exchanged according to mental models, but these are more likely to be developed as part of the PDES effort itself. The mental models will originate at the Application Layer and will be "standardized". Examples are a Mechanical Products mental model, an Electrical Products mental model, an Architecture, Engineering & Construction (AEC) mental model, etc. Each mental model will contain entities and information specific to the particular discipline of the mental model. In fact, we will most times use the term "discipline model" instead of the term mental model.

Drawing again upon the IGES experience for a moment, we see that the exchange of drawings is oriented toward human interpretation of the exchanged data. Therefore, in the data exchange process, it is permissable that the exchange of the

5

mental model be at the human level. Reflecting for a moment, we realize that the exchange of this mental model actually takes place when we learn how to "read" a drawing according to a certain set of conventions.

The situation with PDES is different. Product data expressed with PDES is supposed to be able to be interpreted and used by another computer. This means that the mental model must be computer readable, must be able to be made explicit in the data, and must be able to be exchanged with its structure intact.

Since PDES intends to address all of product data, this means that the totality of all discipline-specific entities in all discipline models is potentially unbounded.

What are the possible implications of this as far as the PDES specification itself is concerned?

One possibility would be to simply have the set of PDES entities be the union of all discipline-specific entities found in all discipline models, and to let this be as dynamic a collection as required.

However, most CAD/CAM systems today, and probably for some time to come, are predicated upon small, stable sets of "generic" context-independent entities (eg., line, arc, string, group, etc.), rather than upon a (user-definable) set of discipline-specific entities that is open-ended. This is particularly true of systems that strive to be inter-disciplinary.

Therefore, these CAD/CAM systems will expect to interact with PDES product data by means of a small, relatively fixed "core" set of generic entities. This is the essential deterrent to simply having the PDES entity set be a dynamic collection of discipline-specific entities.

This simultaneous need to be able to handle an open-ended number of discipline-specific entities and also to be able to communicate product data according to the PDES specification through a small, stable set of generic entities is what leads to the notions of integration and conceptualization. Integration is the determination of the rationale behind expressing a divergent set of discipline-specific entities in terms of one set of generic entities. Conceptualization is the process of abstracting across the different disciplines for the purpose of arriving at a sufficient, non-redundant set of generic entities. Conceptualization also occurs within a discipline as common entities and structures are discovered that support applications within a discipline. It should be noted here that the notion of conceptualization will also be used in a broader sense in this report to mean the act of discovering and formalizing meaning.

6

It turns out that abstraction is a major component of conceptualization.

Redundancy in the generic entity set is undesirable because it opens the situation to non-uniqueness: the discipline-specific entities could then be expressed in terms of the generic entities in different ways. (The notion of expressing a standardized set of discipline-specific entities in terms of generic entities is, historically, not an explicit part of the IGES effort, although some work toward making these kinds of correspondences explicit does appear in IGES Version 3.0. There is, however, redundancy in the IGES entity set, and this does lead to non-uniqueness. The lack of explicit correspondences further complicates matters. The term "flavoring" has been coined to apply to this situation. We say that IGES files are "flavored" according to the user, the system, the application, etc., rather than expressing certain standard information in a certain standard form.)

Suppose now that we do relegate the development of the various discipline models to the Application Layer, and the development of the generic models to the Logical Layer. Analogies have been given within the PDES community to further explain the contents of the Logical Layer, and the relation between these two layers:

The schema for the generic entities (the Logical Layer Model) is akin to the Periodic Table of the Elements in Chemistry. Just as other compounds are described in terms of the atomic elements in the Table, so too are discipline-specific entities described in terms of the generic data described by the schema within the Logical Layer. With respect to the relation between the two layers, we can think for a moment about cooking. The Logical Layer consists of data about flour, leavening agents, spices, etc. The application Layer contains receipes which combine various subsets of the Logical Layer ingredients to produce cakes, cookies, breads, etc.

From the perspective of the Logical Layer, we believe then that a data exchange situation in the PDES context will involve the following three elements:

1) a discipline model that provides the context for interpreting the data for a specific purpose

2) the set of generic entities

3) the set of correspondences from the discipline-specific entities in the discipline model to the generic entities

7

For each discipline model, these three elements form a "data exchange unit". This unit must be computer interpretable, and must be able to be made explicit in the data to be exchanged. Stated more precisely, a data exchange unit should be considered as a three part schema, and the physical file data should consist of instances of this schema. (Note that items (1) and (3) are missing in the IGES environment, as was mentioned above.)

Clearly, it will happen that the same generic entity will be used in different ways by different entities within the discipline models, possibly even within the same data exchange unit.

Data exchange out of PDES then will involve consistent actions applied to the entire data exchange unit. On the basis of the common sharing of the discipline model, the receiver will define his correspondences from the discipline model to his own generic entities. From this, the required translator actions from the PDES generic entities to the recipient's generic entities can be determined.

By taking the correspondence from the PDES discipline-specific entities to the PDES generic entities into account, the translator action can be based on information at the Application Layer as well as the Logical Layer. Current IGES translators, having no correspondences to draw upon, must determine their actions based solely on generic type information. These actions are then fixed, either permanently, or at least for an entire exchange set of data.

We can observe how this notion of data exchange units permits a clear separation of the Application Layer from the Logical Layer.

1.5.2   Requirements Imposed On Users And Vendors By PDES

Assume for the moment that product data in PDES is to be expressed in terms of data exchange units as described above. What are the requirements on users entering data into their own systems, and on translators, that must be met in order to be able to establish and use the data exchange units in PDES?

We believe the establishing of data exchange units can be done only when a user's product data is generated according to the same units. For this, extensive analysis by users of their own application areas in terms of the standard PDES discipline models will be required, as will strict adherence to standards and conventions in entering the data into the computer systems. Users must set up their own correspondences between the discipline models and their own set of generic entities, and then must religiously abide by these correspondences when entering data into their systems.

8

A new generation of translators will be required. Users will have to be able to specify the translator actions from the generic entities of their own system to the generic entities of PDES on basis of what their own correspondences are between the discipline model and the generic entities of their system. Similar requirements on translators for taking data out of PDES were given above.

1.5.3 The Data Exchange Capability Represented By The Initiation Work

Our initiation work has not addressed the operational aspects of data exchange. It has addressed only the development of the PDES specification. As a result, our work has not resulted in a complete data exchange capability. What our work has resulted in, however, is a specification of PDES content and a partial data exchange capability. The data exchange capability is incomplete because we do not at this time have a formal and computer readable way of describing mappings between the Discipline models and the generic entities of the Logical Layer Model. This incomplete data exchange capability has, however, been sufficient to demonstrate and verify the capability of a physical file format.

The Logical Layer initiation effort has basically addressed the development of three kinds of models:

1. Discipline Models
2. A Logical Layer Model
3. Global Models

A Discipline Model represents an application experts view of a discipline area such as flat-plate design. All objects and structures in this model will be created from the view point of the application expert.

The Logical Layer Model is the summation of the resource models: geometry, topology, presentation, and geometry-topology associativities and any generic structures involving relationships across resource models. Resource models contain only generic entities and structures that are common to many application areas. The Logical Layer Model contains no discipline-specific entities. The resource models are described in appendix C1, C2, C7 and C8.

The Global Model is a composite model that contains both discipline-specific and generic entities. The Global Model is a convenient informal mechanism for expressing how objects in a Discipline Model are replaced by generic objects from the Logical Layer Model. This replacement scheme is referred to as a "correspondence" in this report. The purpose of the Global Model in the initiation effort is to informally describe the

9

correspondence between each Discipline model and the Logical Layer Model. This description is then used to assist application experts in validating the correctness of the correspondence. The Global Model does not represent an exchange unit as described in section 1.5.3. This model is used in the initiation effort to prototype an exchange content for the purposes of validating the physical file format.

We have not completed the definition of the data exchange units as described earlier. Specifically, we do not at this time have a formal technique for describing the mappings from the Discipline Models to the Logical Layer Models or their inverses. In lieu of this capability, the Global Model stands as an informal exposition of correspondence between Discipline and Logical Layer models. We believe the Global model has an economy of visual expression that makes it easy to read. We believe that this kind of informal expression is a natural precursor to more formal techniques of mapping and validating and, in fact, we may find it necessary to continue to use informal expositional forms in conjunction with formal methods as aides for human understanding of the specification. Ideally, the mappings determined during the development of the specification in follow on PDES efforts would be the mappings used in establising the data exchange units in an actual data exchange situation.

We have provided a specific example from the Electrical discipline to further clarify the difference between the Data Exchange Unit and the Global Model. Figure 1-6 pictures a portion of a data Exchange Unit showing that a structure within the Electrical Discipline Model consisting of the discipline objects "connection geometry", "start place", "end place", and "symbol connection place" is <u>replaced</u> by a generic structure consisting of the generic objects "Composite segment2", "Curve segment2", "Place2" from the geometry resource model; "Edge", ""Vertex" from the topology resource model; and "Vertex-place", "Edge-curve" from the geometry-topology associativity resource model.

The Global Model described in figure 1-7 is a "smashed" version of the data exchange unit of figure 1-6. (This model is described in more detail in figure 4-7.) The correspondences between the Electrical discipline model and the Logical Layer Model are not clear unless one examines the Discipline Model and the Global Model together. Doing this one can informally deduce the correspondences and can express them in a set of natural language "replacement" sentences as we did in the previous paragraph. It is conceivable that a Global Model could consist entirely of Logical Layer entities as a result of a complete substitution of generic entities for discipline specific entities. As we move closer to this "pure" Global Model with no discipline flavor, it will become harder to deduce the

10

Symbol instance

Connection geometry

Start Place

End Place

Symbol connection place

Network

APPLICATION ENTITIES

Edge Curve

Composite Segment 2

Edge

Curve Segment

Vertex

Place 2

Vertex place

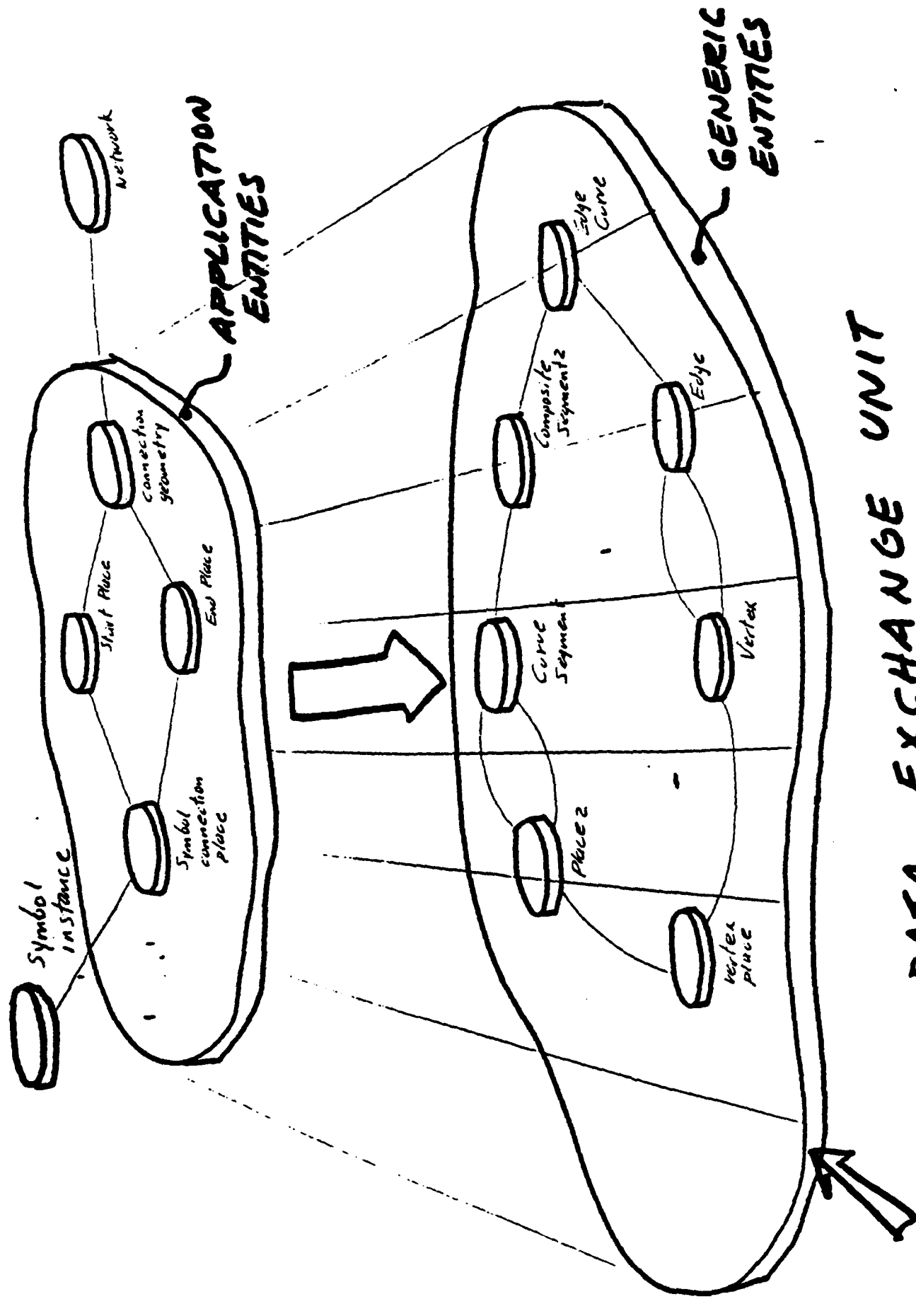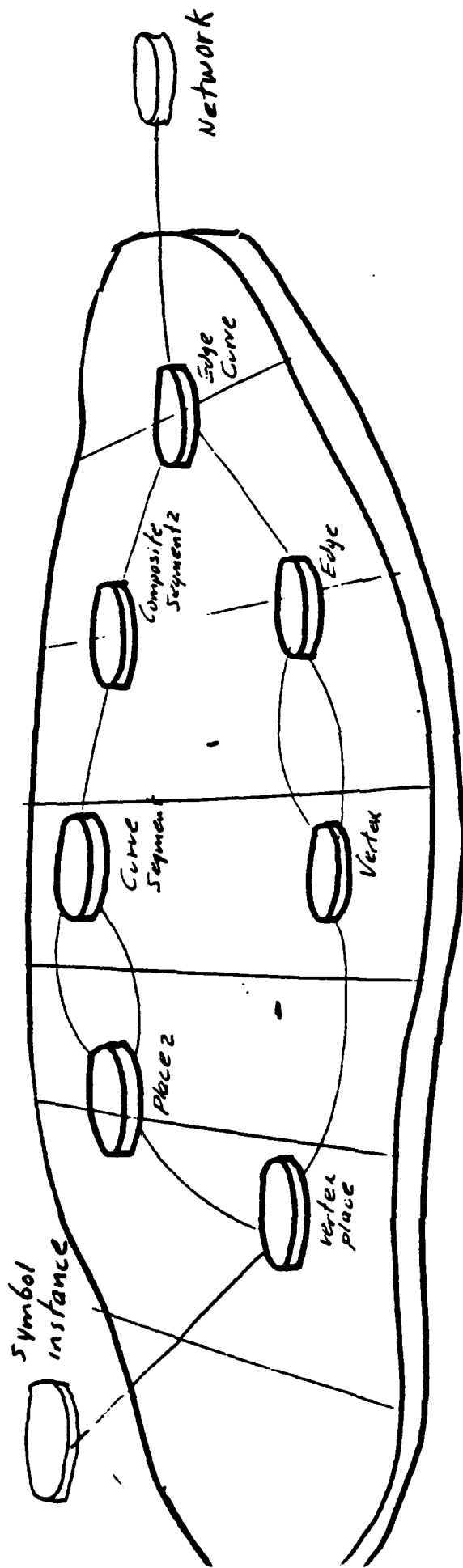GENERIC ENTITIES

DATA EXCHANGE UNIT

figure 1-6

Logical Layer No. 1

GLOBAL MODEL

figure 1-7

correspondences and the need for formal descriptions of mappings will become critical. We did not reach this critical point in the initiation effort.

If we were to use the Global Model in a hypothetical data exchange situation, the interpretation of the exchanged data would then involve a recovery process which would convert from the generic entities to the discipline-specific information that is carried by the generic entities. Thus, the recovery process itself turns out to be dependent upon a (mental) model that is not present in the data. Recall from section 1.5.1 that the inclusion of the mental model in the physical exhchange data distinguishes the PDES from IGES; we have fallen short of the mark by not having a complete mental model in the initation effort.

One shortcoming in this situation (of not having implemented the complete mental model that would be mapped from the discipline model to the Logical Layer explicitly) is that the recovery process is expressed in terms of generic entities, and must apply to the entire set of exchange data. For example, if a certain type of spline "configuration" is to be interpreted as a certain part feature, then that configuration must have that interpretation for the entire data set. If splines are to be used also for something else, and are therefore to be interpreted differently, then something in the spline "configuration" must be different. For example, it could be the case that a spline on one level (i.e., one spline configuration) could be interpreted with one meaning, while a spline on another level (i.e., another spline configuration) could be interpreted with another meaning.

In the Electrical Global model (refer to figure 4-7) the generic associativity "vertex place" in some instances plays the discipline role of "Intersection place" and in other instances the discipline role of "Symbol connection place."

We can observe how, with data exchange units, in which the mapping is in the other direction, this situation is avoided as a result of the physical file data containing instances of the data exchange unit schema.

Another shortcoming of the situation, that is, of not having implemented the mapping from the discipline model to the Logical Layer Model explicitly in the data, is that the clear separation between the Application Layer and the Logical Layer is lost. The Global Model of the Logical Layer becomes a mixture of application-specific entities that have not been broken down into generic entities, and generic entities that have replaced certain application-specific entities.

11

1.5.4 Summary And Conclusion

This section has distinguished between writing a specification for a data exchange standard, and actually using that standard to exchange data. The latter notion is more encompassing in that it also involves user data bases and vendor translators. The initiation effort has not involved actual data exchange.

This section has also introduced the notion of a data exchange unit in the belief that data exchange in the PDES context must occur in terms of these units.

Finally, this section has indicated what data exchange capability our present initiation work would afford. It is seen that the present capability falls short of the more ideal situation in which data exchange units are handled.

It must be pointed out that these current thoughts concerning what data exchange in a PDES environment might eventually involve are certainly not yet seasoned thoughts, and may very easily change as we continue to learn in connection with the longer term PDES effort. We have not yet had the opportunity to thoroughly discuss them with others within the data exchange community.

However, we can use these thoughts on these topics to illustrate that there exist many fundamental issues related to the PDES effort that are just now surfacing. These issues have yet to be thoroughly discusses or resolved.

There must be a clear understanding that, while the PDES Project has signed up for certain delivery dates for Version 1.0, the mere setting of these dates is not to be interpreted as an indication that all work steps necessary for achieving them and for the successful use of Version 1.0 are well understood in terms of present CAD/CAM practices, and that all that remains is simply to routinely carry them out.

In fact, it is our recommendation that the PDES effort ought not be driven primarily by schedules, but rather by an intention to take the time to formulate, analyze, and solve the substantive issues that accompany the approach to developing the PDES specification according to the Second PDES Report.

2.0 The Logical Layer Methodology

The purpose of the PDES Initiation Logical Layer Methodology is to provide a framework in which to guide the development of a single logical model that will support all applications within PDES. This model will be called the Logical Layer Model.

The methodology described here will:

1. Maximize the usage of human resources by defining and establishing project roles

2. Support the development of an informal correspondence (mapping) between the Logical Layer Model and each Discipline Model

3. Maximize the potential of the Logical Layer Model and mappings to and from it to serve as a central resource from which all other PDES forms can be derived

2.1 Cognitive And Specification Models

The application and logical layer both are responsible for formally describing meaning. The application layer describes meaning within a relatively narrow subject area. The logical layer describes meaning common to many subject areas.

We define conceptualization as the act of discovering and formalizing meaning. This definition holds regardless of the size of the subject area or the number of subject areas. By this definition, both the application and logical layers involve conceptualization.

Cognition is the component of the discovery and concept forming process that emphasizes human perception. Any modeling technique which has as its principal objective the promotion of human perception of concepts (any concept) will be referred to as a cognitive modeling technique. The result of such a technique will be referred to as a cogntive model. In this report we will use the terms "cognitive model" and "conceptual model" interchangeably.

Abstraction is a major factor in the concept forming process. It is a process whereby humans express qualities or characteristics of an object, event, or phenomenon apart from any specific instance of that object, event, or phenomenon. Cognitive modeling techniques promote the abstraction process by providing simple means (usually graphical) for expressing objects, events, and phenomena and their interrelationships.

13

Specification is the process of packaging and reporting on the results of conceptualization. Any modeling technique which has as its principal objective the textual packaging of conceptual models will be referred to as a specification modeling technique. Specification is not part of the conceptual-model building process. No new concepts or relationships will be discovered during specification, however, specification will result in the repackaging of the conceptual model to make it more concise and perhaps more readable. Repackaging involves grouping, concept typing, and naming.

Cognitive models normally have a strong graphic component and a weak text component. Graphics promotes human perception through "picturing" (picturing is not the only means of promoting human perception, however). Specification models are purely textual and structured into an "entity" form. The cognitive model provides a clay-like medium in which the modeler can mold concepts. The specification provides a casting medium to display the results of conceptualization. In the initiation effort these are distinct models.

These two models must have equivalent descriptive power and precision and they must share a common foundation of basic notions such as: object, relationship, proposition, class, and function (the basic notions of first order logic). Both models must have a computer sensible representation. Each must be derivable from the other.

It is recommended for PDES follow-on work that congnition and specification be recoginized in the Logical Layer Methodology as distinct processes accomplished separately but in a coordinated manner. It is our opinion that any attempt to combine these two processes into a single process will degrade the quality of the resulting models. It follows then, that the use of the Logical Layer Methodology should result in two distinct models: a cognitive model and a specification model.

In the initiation effort, the NIAM irreducible binary model was selected as the standard cognitive model. More exactly, this modeling technique is referred to as the MIML subset of the Nijssen Information Analysis Method (NIAM). (See Reference 2.) This subset is a semantic-net language capabable of declaring object types, object type names, object roles, binary relationship types, object subtypes, and advanced constraints. The MIML capability is a subset of the full NIAM modeling capability. (The full NIAM constraint capability, referred to as Reference and Idea Language RIDL, includes procedural constraints and advanced declarative constraints.) Hereafter, in this report, references to NIAM will mean the MIML subset of NIAM. NIAM is described in a primer in Appendix D2.

Basically, NIAM is a weakly-typed semantic-net language which does not force grouping of concepts. However, almost any kind of grouping (packaging) can be superimposed on models developed with NIAM.

DSL was selected for the initiation effort as the standard specification model. DSL is a variant of the language used to specify the PDDI conceptual schema. This language is described in a primer in Appendix D3.

Basically DSL is a strongly-typed linear language for packaging the objects in conceptual models into groups called "entities". DSL has the advantage that it does not force any particular grouping style on the NIAM models (such as the popular data base normal forms). Grouping is the critical link between the cognitive model and the specification model. The basic organizational mode of DSL is the nested array. Nesting allows more concise and natural descriptions of PDES artifacts. DSL does not demand that the cognitive model be nested, however.

No single modeling technique was stipulated for the application layer. IDEF-1 was the technique of choice in several areas. (See the primer in Appendix D1.) Some preferred to model directly using specification-like text models similar to DSL.

2.2 Overview Of The Methodology

The methodology is illustrated in Figure 2-1. It breaks the development of models into phases, defines interfaces between phases, and assigns project roles. These are the phases:

Phase 1: Qualification

Phase 2: Global Conceptualization and Integration

Phase 3: Specification

The input to the methodology is a set of discipline models. In the initiation effort these models include the Flat Plate Mechanical Part, Finite Element Modeling, Electrical Schematic, and Tolerancing application area models.

15

METHODOLOGY ARCHITECTURE

Application

Logical

Phase 1
Reduce to Binary
and Quality

Phase 2
Build Global Model

Phase 3
Group and Specify

Physical

Discipline 1	Discipline 2	Discipline N
2 Reference Model	**2** Reference Model	**2** Reference Model

Reduce

Verify

3 Qualified Discipline Model (NIAM)

4 Global Model (NIAM)

5 Specification (DSL)

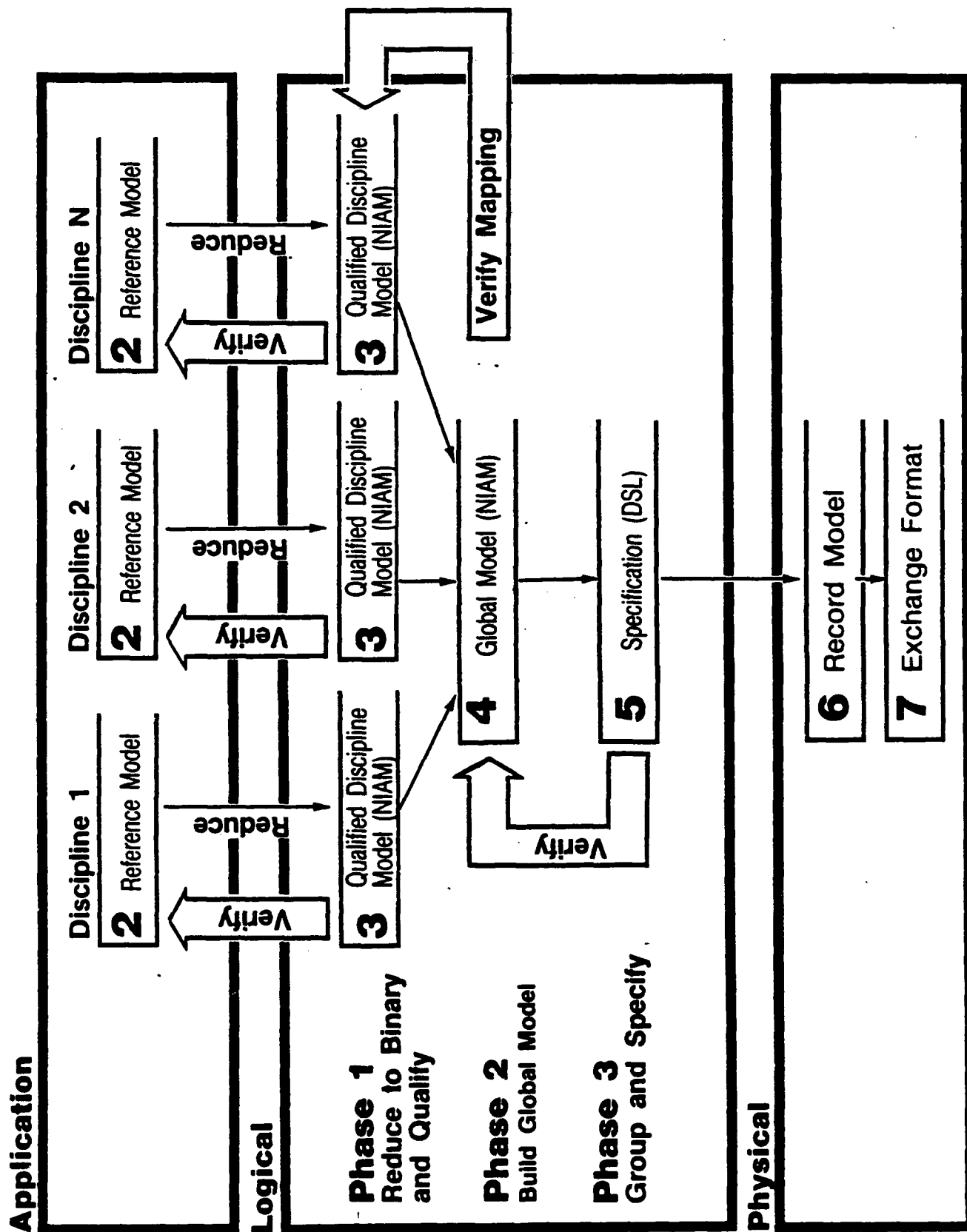Verify Mapping

6 Record Model

7 Exchange Format

FIGURE 2-1

The ultimate deliverables of the methodology are as follows:

1. All of the Discipline models in a standard cognitive model form

2. A Global Model for each Discipline Model in a standard cognitive model form

3. A map between each Discipline Model and its Global Model (this was not formally accomplished in the initiation effort)

4. A Specification Model of each Global Model in a standard specification form

It should be emphasized that the methodology is independent of any particular modeling technique (with the exception of its cognition, specification cateogry). The methodology should be viewed as a broad framework stipulating two modeling forms, the cognitive form and the specification form.

2.3 Description Of The Methodology

Phase 1: Qualification

Goal: To maximize the global conceptualization and integration potential of the discipline models

Input: Discipline models (in any modeling language)

Deliverable: Discipline models (in a standard cognitive form)

This phase actually represents an interface to the Application Layer and insures that all discipline models appear in a uniform language that maximizes the chances for successful global conceptualization and integration. The assumption we are making at this time is that no standard Application Layer modeling language will be stipulated. This does not prevent the application layer from selecting the standard cognitive model as its modeling choice. If the cognitive language is powerful enough to express the global abstractions (concepts) of the logical layer it is powerful enough to express the discipline abstractions (concepts) at the application layer. This phase thus allows for the fact that modeling languages used at the Application Layer may not be the best languages for the extensive conceptualization that must be done at the Logical Layer.

In summary, this phase buffers the Logical Layer from the variety of languages at the Application Layer, and also paves the way for the extended conceptualization that must occur in the Logical Layer.

16

A discipline model is said to be "qualified" when it is determined that the discipline model in standard form is equivalent to the discipline model in native form. Ideally this phase of the methodology should be concerned with translating the Discipline model to a standard cognitive form. Any validation done in this phase should be for the purposes of insuring that the meaning of the Discipline model has been preserved in translation to the cognitive form.

It is not the responsibility of the Logical Layer to perform discipline modeling, however, they do have the responsibility to read and understand the discipline models and request clarification or correction to the models if needed.

Phase 1 roles identified:

1. Model Translator - This person must be familiar with the particular discipline modeling language and the standard conceptual modeling language of the Logical Layer.

2. Discipline Liaison - This role requires that the participant be familiar with the discipline and the particular modeling language of the discipline. It would be ideal if the participant were also familiar with the Logical Layer standard modeling language. This is an extremely demanding role.

Phase 2:   Global Conceptualization and Integration

Goal: To discover and formally model generic objects, events, and phenomena (referred to as resources) that are common to all disciplines and determine how they can be used to replace discipline specific objects, events, and phenoema.

Input: A Qualified model of each discipline

Deliverable: 1. A set of resource models containing only generic objects, events, and phenomena (the set of all resource models is referred to as the Logical Layer Model)

2. A Global model that formally describes how each Discipline model is represented in terms of the generic objects, events, and phenomena of the resource models. The union of these Global models represents the model universe of PDES.

17

This phase is the most challenging of the three phases and is the heart of the methodology. The Logical Layer team believes that the discovery and formalization of generic objects, events, and phenomena across the diverse disciplines within PDES is a task that has never been accomplished before. Admittedly, this phase of the methodology will evolve over time as we become more mature in conceptualizing over diverse application areas. This kind of conceptualization will require broad knowledge of many application areas on the part of logical layer workers in addition to mature modeling skills. The ability to abstract very general structures and semantic classes will be critical. It is felt that this extreme form of abstraction goes beyond the usual notion of abstraction that is attached to the process of conceptualization of more restricted domains of discourse.

A set of baseline tasks are suggested for beginning the Phase 2 global conceptualization effort. Evolution to more mature techniques of global conceputalization would be expected to occur from this baseline.

Baseline tasks:

1. Develop a set of concept categories. To the greatest extent possible, these categories will be generic in that they may be used across a broad set of disciplines. Ideally these categories can be discovered as the discipline models are being qualified in Phase 1 (bottom up), or they may need to be adopted provisionally on the basis of general knowledge derived from other sources such as IGES, PDDI, PHIGS, etc. (The latter was the case in the initiation work with the resource models of wireframe geometry, topology, and presentation.)

2. Develop a set of structural categories. A structure in this case is a recognizable pattern of interrelated concepts that appear in multiple discipline models. (We assume when referring to a discipline model that it is qualified.)

3. Examine each discipline model and attempt to factor it completely into generic concepts and structures and discipline-specific concepts and structures. Denote generic concepts and structures so they can be clearly identified. The resulting model will be referred to as the Global model in this methodology. Generic concepts in the Global model are denoted by containing them within a closed dotted line.

4. Once the discipline model has been converted to Global-model form, logical layer workers in cooperation with the application liaison person verify that the qualified discipline model can be recovered from the Global model. They should record this recovery process in a formal language. If no formal mapping

18

language is available, the recovery process should be recorded in sentence form. It should be emphasized here that the main purpose of the Global model is to provide a convenient exposition for showing the correspondence (mapping) between a discipline model and its corresponding Global model. It is not to be considered an exchange model per se. Please relate this section with the discussions in section 1.5.3. In the initiation effort, the Global model is the only mechanism for showing the correspondence between a discipline model and its corresponding Global model. We did not develop a formal mapping language in the initiation effort. Some mappings, however, are expressed in natural language in this report.

5. Any new generic concepts or structures that may have been discovered during the conversion of discipline models to Global models should be formally noted. Previous discipline models should be reviewed again to see if any of the new generic concepts or structures can replace discipline specific concepts or structures in those models.

In summary, the process is heuristic. In the initiation effort a baseline of entities known to be generic (geometry and presentation) was first modeled into a Task 1 Logical Layer Model in order to get the project off the ground. Ideally all generic entities would be discovered in the process of discipline modeling. It is difficult to establish a recipe for global model building. Basically, this phase keeps substituting and/or expanding concepts and structures in the discipline models in terms of generic concepts and structures to the maximum extent.

The summation of all the Global models constitutes the conceptual-model universe for PDES. The collection of all qualified discipline models, the correspondences (mappings) between the Global model and the qualified discipline models, and the summation of all the Global models will be referred to as the Integrated model.

Integration is defined in this methodology as the act of creating the Integrated model.

Phase 2 roles identified:

1. Conceptual modeler with broad discipline knowledge

2. Discipline liaison

19

Phase 3: Specification

Goal: To package each Global model into a concise, aggregated (entity-based) form.

Input: Global models

Deliverable: A specification model for each Global model

This phase involves the translation of each Global model to some standard text form called "the specification". Ideally, this phase would not involve additional conceptualization. The translation to the specification should not delete detail (although it may abstract out detail through the strong typing capability of the specification language) or add new meaning (except that which may be assigned to groups and types). The principle difference between the specification and the Global model is that some form of grouping has been applied in order to make the Global model more compact. Grouping is done based on the concepts and relationships present in the global cognitive model and is the critical link between the Global model and the specification model. Grouping is done for the purposes of achieving compactness and readability not for the purposes of achieving further conceptualization. Grouping is not done for the purposes of achieving a logical record design in preparation for implementation design. The specification is text only, whereas the global model can be expected to be a mixture of graphics and text.

Ideally, the translation from the global model to the specification model would be automated. It was a manual process in the initiation effort. Since the methodology just described does not force any particular model or specification language, it cannot provide the details of translation. However, our experience in the Initiation Effort indicates that the development of the global modeling language and the specification modeling language should be a coordinated effort to insure that syntactical constructs in one have equivalent constructs in the other.

Likewise, the methodology does not provide a technique for grouping of global models. In the initiation effort grouping was a joint effort between conceptual modelers and specifiers.

It should be emphasized that the resulting specification models represent, in a concise, textual form, the correspondences between discipline models and their Global model counterparts. The specification does not represent an exchange mechanism per se. However, in the initiation effort, since this is the mechanism for transmitting PDES information content to the physcial layer; the specification serves the role of being an exemplary exchange for the purposes of testing the physical file

20

format. It is recognized that the exchange unit as described in section 1.5.1 will ultimately replace the Global model as the principal mechanism for representing the correspondence between discipline models and the generic objects in the Logical Layer Model. The input to this phase would then be logical exchange units instead of Global models.

Phase 3 roles identified:

1. Specifier - This person is responsible for creating the specification. This person coordinates with a Logical Layer person to perform grouping and translation.

2. Logical Layer Liaison - This person works with the specifier to coordinate the grouping of the global model. This grouping should be a coordinated effort as the groupings are the main link between the Global model and the specification.

2.4 Choice Of Modeling Techniques

This section discusses the selection of modeling techniques for the three phases of the methodology.

For Phase 1 (Qualification) and Phase 2 (Global Conceptualization and Integration), a cognitive language was chosen because human perception and abstraction are the principal activities involved in these two phases. For Phase 3 (Specification), a specification language was chosen.

The logical layer had no jurisdiction over selection of languages to be used at the application layer. For the most part, discipline models were modeled in IDEF-1.

Cognitive Language Choice

For Phase 1 and 2, the Nijssen Information Analysis Method (NIAM) was chosen as the standard modeling technique. NIAM is a semantic modeling technique. It emphasizes elementary concepts and binary relationships. The following is a summary of NIAM strengths:

1. It has international recognition.

2. The method is the public domain (MIML subset).

3. There is a body of people with experience in the methodology on several continents.

4. It is oriented towards a sub set of natural language.

5. It has a rigorous formal basis in classical and linguistics and logic.

6. It has a formal grammar (BNF).

7. It is a purely semantic technique and involves no grouping.

8. It is detail exposing.

9. It is a bridge to advanced knowledge representation techniques.

10. It has strong constraint representation capability.

11. Computerized tools are commercially available to support model creation and access, dictionary management, and graphical output.

We emphasize again that Phase 1 and 2 are cognitive activities. In these phases, nothing should interfere with the processes of perception, concept discovery, or the exposition of detail. We specifically `~ against modeling techniques which attempt to combine conceptualization and grouping or conceptualization and specification. We believe these techniques will not yield the high quality models needed for integration.

We have specifically not chosen the entity-attribute-relationship type of model because it mixes the conceptualization and grouping processes, it forces the modeler to artificially separate concepts into entities and attributes, it does not sufficiently expose detail, and it does not graphically represent advanced constraints.

IDEF-1 is a hybrid modeling technique that combines aspects of entity-attribute-relationship modeling and relational modeling.

For a formal comparison of the binary approach and the entity-attribute-relationship approach, the reader is referred to Reference 3, ISO/TC97/SC21/WG5-3 report SC21-N197, Appendix D and E.

Specification Language Choice

The Data Specification Language (DSL) was chosen as the standard specification modeling technique for the initiation effort. It has the following strengths:

1. It is compact and purely text.

2. It has a text structure that is easy to read.

3. It is strongly typed to support abstraction.

4. It has a formal grammar (BNF).

5. It has a close relationship to the binary model.

6. It has potential for addition of advanced constraints.

7. It has a basic organizational foundation (nested array) that promotes representation of complexity in a natural way

DSL is a derivative of the language used to specify the PDDI conceptual schema. A primer on DSL contained in Appendix D3.

3.0 Application And Resource Areas

This section consists of a brief overview of the major application and resource areas used in the initiation effort.

On the one hand, given that the main emphasis in the initiation effort is on methodology, the use of these content areas and their corresponding models is strictly as a means to an end. On the other hand, the content itself does deserve exposition. It was developed using the modeling approach, and the people involved represent a fairly broad cross-section of standing IGES technical subcommittees; specifically, the Curves and Surfaces subcommittee, the Finite Element Modeling subcommittee, the Electrical Applications subcommittee, and, in the case of the Tolerancing application model, the Chairpersons of the Manufacturing Technology subcommittee, the Drafting subcommittee, and the Mechanical Products subcommittee.

Recall that resource models describe the entities used as generic entities. Global models then describe discipline specific entities in terms of these resource entities.

Each overview statement below was written either by the task leader of the area being described or by a person intimately involved with the development of the model for that area.

3.1 Wireframe Geometry Resource Area - Edward Clapp, IBM - Chairman, IGES Curves And Surfaces Subcommittee

The Wireframe Geometry Proposal is intended to meet the PDES geometric needs for points and curves. Among its design criteria were:
- ability to meet the geometric needs of CAD systems;
- ability to meet the geometric needs of Numerical Control;
- ability to be extended in a uniform manner to surfaces;
- high priority to meet the needs of Boundary Representation Solid Modeling;
- ability to symbolically represent much of the information;
- minimal entity set with simple and uniform definitions;

- stable representations for the geometry.

The primitive data types are
- reals;
- integers;
- booleans;
- character strings (so as to be able to refer to significant information by name rather than by value).

The basic building blocks are arrays. 2 dimensional and 3 dimensional geometry have been separated. Consequently the most commonly used arrays are ordered pairs and ordered triples. These are used for position and direction. Ordered lists are also used for the composite curve.

The geometry itself consists of points and curve segments. A point is defined in terms of its position in space, either by name or by value. A curve segment is defined in terms of its underlying curves, endpoints, and its direction. The curve and end points may be defined either explicitly or referred to by name. This reference by name allows for symbolic statements about intended continuity conditions.

The curves (both 2D and 3D) are
- line;
- circle;
- ellipse;
- hyperbola;
- parabola;
- nonuniform rational B-spline;
- composite curve.

Deficiencies:
- offset curves were omitted;
- geometric tolerancing, while present, is inadequate;
- no means to allow for user defined parametrization;
- parametric evaluation capabilities need to be fit into this schema;
- a certain amount of apparent complexity has been introduced, which perhaps can be resolved at the DSL and/or file structure level.

It must be noted that this approach has generated some controversy. There are a number of entries in the Curve and Surface document list dealing with this, including two alternate proposals.

3.2 Presentation Resource Area - Richard C. Winfrey, Digital Equipment Corporation - Member, IGES General Assembly

PDES is intended to allow the exchange of all data necessary for the manufacture of a product. Here, manufacturing is taken in its broadest sense. Strictly speaking, the specification of product data could conceivably be done without making allowances

for displaying that data. In this sense, presentation data is not a PDES requirement. However, it was felt that information required to reconstruct the same presentation (eg., graphics display for now, possibly engineering drawing for later) of product data on both a sending and a receiving system was an important capability and should be included as part of the PDES specification. In addition, it was realized that in the existing IGES specification, the data defining certain entities was intertwined with display attribute data for those entities. The approach to avoiding a repetition of this undesirable situation was to include presentation as a work item in the PDES start-up activity, i.e., the PDES initiation effort, and in that work item insure that the two kinds of data did not get intertwined.

In defining the Presentation functionality, it was important to decide whether just a picture, or the information necessary to CREATE the picture, should be included. The exchange of just the picture would be relatively simple, and could be done by specifying an existing standard for a meta-file such as found in GKS(2D). However, it was recognized that the need is to exchange information to create the display. The display of a part could thus be created from the actual geometry exchanged rather than from an independent and separate list. In addition, the receiving system would subsequently be able to manipulate the view of the displayed part, and to perform editing functions on it.

Having made the above decision, and after evaluating the functionality found in IGES, it was obvious that what was needed was a subset of a graphics standard. This subset would provide for view operations (i.e., a 3D viewing pipeline) and a display capability for geometry and text. Because of these rather limited needs, any one of CORE, GKS-3D, or PHIGS would suffice for defining the viewing pipeline. However, it was felt that PHIGS had a clear advantage over the others because of its use of a hierarchically structured display list.

A PDES Presentation draft specification was prepared in mid-1985 based primarily on the then-current PHIGS specification. The one deviation from PHIGS was in the area of text, where the CG-VDI specification was followed.

The current status of the PDES Presentation effort is as follows:

1.   The Presentation draft proposal is currently the subject of a mail ballot of the IGES community. Results will presumably be made known at the Raleigh IGES meeting beginning April 28, 1986.

2. The Presentation draft proposal has been formally
   submitted to the ANSI X3H3 committee (PHIGS) along
   with a list of requested modifications to the PHIGS
   specification. It has received a preliminary review
   by that group, and while some differences were
   expressed, they were very willing to work with us to
   better understand our needs and to resolve those
   differences.

3. The ISO TC184/SC4/WG1 group has recently rejected the
   Presentation draft proposal in favor of GKS-2D. As
   discussed above, this would limit the PDES
   Presentation functionality to that of exchanging
   pictures only, since it would then be impossible to
   recover 3D information at the receiving sytem.

3.3 Topology Resource Area - IGES ESP Committee

A provisional topology model was added approximately midway
through the initiation effort. It was added because no
siginificant amount of global conceptualization could be
accomplished without it. The model developed is a pure topology
model in that it does not refer to geometry. The model is
basically derived from the IGES ESP work. The model was developed
directly in NIAM by the logical layer and contains the
topological notions vertex, edge, loop, face, and shell. There
is no claim for this to be the "real" PDES offering for topology.
The need for the model surfaced while performing global
conceptualization of the tolerance model. The tolerance
application group actually supplied their own topology model (in
PASCAL-like syntax).

Some may consider that the use of a full-blown topology model to
overkill for modeling simple connectivity. The problem was that
simple connectivity appeared in application models several times,
each time modeled in a different way. We had a desire to
establish a uniform way of representing simple connectivity
across all applications.

The development of the topology model also motivated another
resource model that we have called GTA (geometry-topology
associativity). Over several months of modeling we have found a
class of relationships between geometry and topology that have
potential for general application. We found numerous occassions
where it did not seem to make sense to associate a particular
fact to geometry by itself or topology by itself. It does not
make sense to paint a topological face nor to paint an unbounded
surface. We have heard such expressions as "topology sits on top
of geometry." We choose to avoid subordinating one to the other
by bringing them together as peers through the GTA

27

associativities. We also had a strong desire to be able to refer to pure topology without any other baggage (such as geometry) being attached.

We realize that we stand in err if "face" and "shell" are not to be interpreted as purely topological objects but, more specifically, as objects of the boundary representation scheme. Our topology model states that a face is a connected region with a boundary defined by a loop. The question is: does it make sense to talk about a connected region without referring to geometry?

In summary we expect topology to be one of the most valuable resources in global conceptualization in future PDES efforts. The initiation effort learning experience more than justified the "stubbing in" of a temporary topology model.

3.4 Electrical Schematic Application Area - Curt Parks, General Dynamics, Pomona Division - Member, Electrical Applications Subcommittee

The electrical schematic is a subset of the information needed to define an electrical product. It was selected for the initiation task because of the number of model entities involved (about 30). These could be worked into a normalized model and a consensus achieved by the Electrical Applications Committee (EAC) within the allocated time period.

The basis of the model was a Printed Wiring Assembly (PWA) model which had been constructed the year prior. The PWA model purpose was to identify the logical information and relationships needed in IGES for Version 3.0 to support the transfer of all PWA CAD data. The schematic entities constituted about half of the PWA model. In turn, the schematic is applicable to all electrical packaging technologies and could form the basis of a broad-scope PDES information set. The schematic is also an early element in an electrical product life-cycle. As such it is used to define the functionality for the product design and for design analysis. Later in the life-cycle, a packaging technology is selected (e.g., a PWA or a Hybrid Microelectronic Assembly). The schematic then becomes a reference information document used when defining product test requirements and logistical support.

In addition to the information entities from the PWA model, the logical development methodology also required a user view and some entity refinement. Both were used to remove implementation-specific data from the model. The resultant model was also converted from IDEF-1 to the IDEF-1 Extended format. Then the model was successively reviewed and refined by the EAC members. When the model was felt to fully support the user view (i.e., could be searched to derive a net list and a part list information set), the relationships and attributes were

28

normalized by entering the model into a "Metamodeler" database at General Dynamics Pomona Division. The database was passed into the JANUS modeler (JANUS is a product owned by D. Appleton Co.). The JANUS diagrams and the Metamodeler listings were used to normalize the model which was drawn on three A-size diagrams on a general purpose graphics design system.

The model was then submitted for integration with the remaining PDES Initiation task application models. During integration reviews, several entities were re-defined. The new entities allowed integration with the geometry entities of the other applications. The resultant model (now on 4 sheets) was reviewed by EAC members. The Metamodeler database was updated and the attributes migrated as required for the final model normalization.

The key conceptual element present in this model has been termed "connectivity". The concept is not limited to electrical; it also defines how a road becomes part of a highway network, or a beam becomes part of a truss assembly. This "functionality" aspect of product definition will be studied further in PDES development.

3.5 Tolerancing Application Area - William C. Burkett,
     McDonnell Aircraft Co. - Chairman, IGES Manufacturing
     Technology Subcommittee

The PDES Tolerance Application Group (TAG) was formed in response to a request by the Logical Layer Initiation Task Group to develop a reference model of the information required to tolerance a product model.

The objective of the modeling effort, as defined by the TAG, was to identify and define the information required to support the tolerancing practices as advocated by the ANSI Y14.5M - 1982 and ISO 1101 and 1660 standards. In addition, the representation of the product was assumed to be a three-dimensional (3-D) geometric model rather than the traditional engineering drawing. Although the above standards are oriented toward dimensioning and tolerancing the pictorial representation of the product (the drawing), it was felt that the concepts presented were equally applicable to a 3-D geometric representation. The TAG, therefore, did not attempt to develop new concepts to tolerance a 3-D geometric model, but rather attempted to capture the concepts presented in the standards and apply them in a new environment. This approach essentially is "a step into the future with a foot firmly planted in the past."

29

More specifically, the scope of the TAG work was defined by the information required to tolerance the shape of a 3-D geometric product model. There are several important assumptions that went into the work:

- The geometric model is a boundary representation solid model or an equivalent surfaced wireframe model. (This is needed to address specific, discrete regions (surfaces) of the product model and the corresponding regions of the physical product.)

- The geometric model represents exactly the nominal shape of the product.

- Dimension information is implicit in the model geometry.

Outside the scope of the TAG work were:

- Computing Tolerances
- Process Tolerances
- Pictorial Representation of the Tolerances
- Dimensioning Practices
- Non-Mechanical Tolerances (e.g. Electrical Component Values)

The reference model produced was based on the work done in this area by the PDDI project. In addition, the work done by R.H. Johnson for CAM-I (D&T Final Report, R-84-GM-02.1) was reviewed. It is felt that the approaches taken by Johnson and that of the TAG are not incompatible, since they differ primarily only in depth and breath.

3.6 **Finite Element Modeling - William R. Freeman, Allied Bendix Aerospace, Kansas City Division - Member, IGES FEM Subcommittee**

The Finite Element Method (FEM)·is a technique for creating mathematical models for engineering analysis. These analyses may include stress, vibration, heat transfer, plastic molding, electrostatics or magnetic fields. The FE method uses a model of the real object divided into regular subdivisions called elements. These elements are defined by points in space called nodes. The nodes and elements, along with additional information such as material properties makes up the model.

The IDEF-1 application model created by the IGES/PDES FEM Committee contains all of the basic information that is contained in a finite element model. The original data required to create that model, such as the shape (geometry) of the real object, the

bill of materials, material properties, and the environmental conditions relating to the analysis (loads, temperatures, etc) comes from outside sources which are not included as a part of the FEM application model.

FEM analysis data were not included in the model. These analysis results are important, but are logically separate from the information required to define the model prior to analysis. This partitioning was intended to match the scope of the application model to the available time and personnel resources.

The application model covers nodes, elements, material properties, environment (loads, restraints, temperature, etc.), grouping of entities, and their interrelationships. Considerable detail is included in the data dictionary, included as a part of the model. The FEM Committee feels that the current application model fairly represents the data relationships necessary to fully understand the information and the information relationships required for a FE model, and is therefore complete. In addition, the model has been verified through the use of the "natural language quality assurance loop" which gives further assurance that the IDEF-1 information model correctly represents relationships between FE information entities.

Future plans include the addition of analysis results to the IDEF-1 model, and some consideration of the sources of original information from which the FE model is abstracted. A real part may indeed be analyzed under multiple environmental conditions, for a number of analysis types, with alternate materials, with various levels of detail resolution, and so on, but it is still the same part, even when represented by many different FE models. At present, this overall concept is difficult to properly place in the order of things, but should be included in the overall electronic data system which PDES is addressing.

In summary, the FEM application model is considered to be complete (with the exception of analysis results), and correct. The IGES FEM Committee has verified and approved this model for a "draft" level of release outside of the PDES working groups. Few if any changes are expected, aside from the extensions for analysis results.

4.0 Details Of Applying The Methodology

4.1 Example 1: Electrical Schematic Application Model

4.1.1 Introduction

This application uses simple 2D geometric constructs, is compact, and exemplifies the principles of the methodology. The purpose of this section is not to review the entire electrical model, but rather to use the model as a vehicle to review an actual use of the methodology.

4.1.2 Review Of Phase 1 Of The Methodology: Qualification

The documentation package as received from the electrical task group is in Appendix C4. This package exemplifies a documentation style which is highly desirable from the vantage point of the Logical Layer. We offer here a condensed version of the document package.

Strategic Directions

SCOPE: This document contains diagrams and text representing a conceptual model of the data required to describe an electrical schematic.

CONTEXT: This model identifies Entities, Relationships and Attributes which are logically necessary to construct an electrical schematic. It also identifies some probable relationships to entities in two other applications: circuit analysis and printed circuit board physical design.

PURPOSE: This model proposes portions of the PDES logical layer required to record an electrical schematic for integration with other portions of the PDES logical layer.

VIEWPOINT: The viewpoint of this model is that of the IGES/PDES electrical subcommittee.

Model Notes for Reviewers

Inasmuch as the intended use of the model is a logical view of the data, independent of existing systems or methods, for development of a neutral product data exchange structure, the reviewer is asked to keep in mind several important modeling constraints:

The entities and attributes are to reflect the data inherent in schematics, as opposed to the information conveyed by a schematic. An example is the user view preceding the logical IDEF-1 model describing a netlist. The netlist is recognized here as information assembled by a query against schematic data. In turn, the model must be capable of supporting such a query. Elements of information found in the netlist are found in the Network, Symbol Connection Place, and Symbol Instance entity classes.

The model must exist independent of implementation. The entity classes in the logical model must be equally valid for a pencil-drawn schematic or a CAE system with model data structures.

The author's opinion is also offered that, strictly speaking, an electrical schematic (the totality of this model) is not part of "product definition". Schematics exist only as a design aid and as reference documentation.

Electrical Schematic Application - User View

I.   Definition

The schematic is a symbolic representation of component parts and their electrical connections. The schematic may apply to any hierarchical level of product definition, and becomes part of the packaging requirements at that level. Schematics may also contain details of related mechanical nature (e.g. heat, sinks, connectors, etc.) and transmission of optical, magnetic or microwave energy.

II   Inputs (sources of design constraints)

o Block diagram, system or subsystem with target block identified

o Interface requirements (e.g., signals and power)

o Mechanical package requirements (e.g., chip, hybrid microelectronic assembly or printed board size and mounting)

o Design (and product) constraints and characteristics (e.g., specifications, system equations, test requirements)

o Schedules and/or budget

o Approved (or preferred) parts lists

o Symbol set and drafting standards

III Items schematic relates to during design

    o Design block diagram

    o Boolean operators

    o Detail equations/transformations

    o Static or dynamic models and simulations

IV Schematic constituents

SYMBOLS: a 2-dimensional figure commonly accepted as a representation for a part's functionality.

SYMBOL CONNECTION POINTS: indicates where connections to symbol can occur.

JUNCTION POINT: Symbol (usually a filled small circle) indicating an electrical union of 2 or more connection lines.

CONNECTION LINES, SINGLE: a 2-D line or string between symbol connection points or junction points.

V Schematic outputs

PICTORIAL: used to review circuit design and as part of final product documentation.

NET LIST: a topology derived by examining the logical relations (links) created by the connection lines (joints).

BILL-OF-MATERIALS: a list of parts cited in schematic. Used stress analysis, packaging the physical circuit and in documentation parts lists.

Glossary

CONNECTION GEOMETRY: Geometry which represents the logic of an electrical joining.

INTERSECTION PLACE: A place where connection lines are considered to be connected.

LINE: A narrow, elongated mark drawn or projected.

NETWORK: A collection of places which are defined to be at the same potential (aka Node).

SCHEMATIC: A symbolic representation of a function.

STRING: A piecewise, continuous form of a 2-D line (aka polyline).

SYMBOL CONNECTION PLACE: The location on a symbol where a connection may occur.

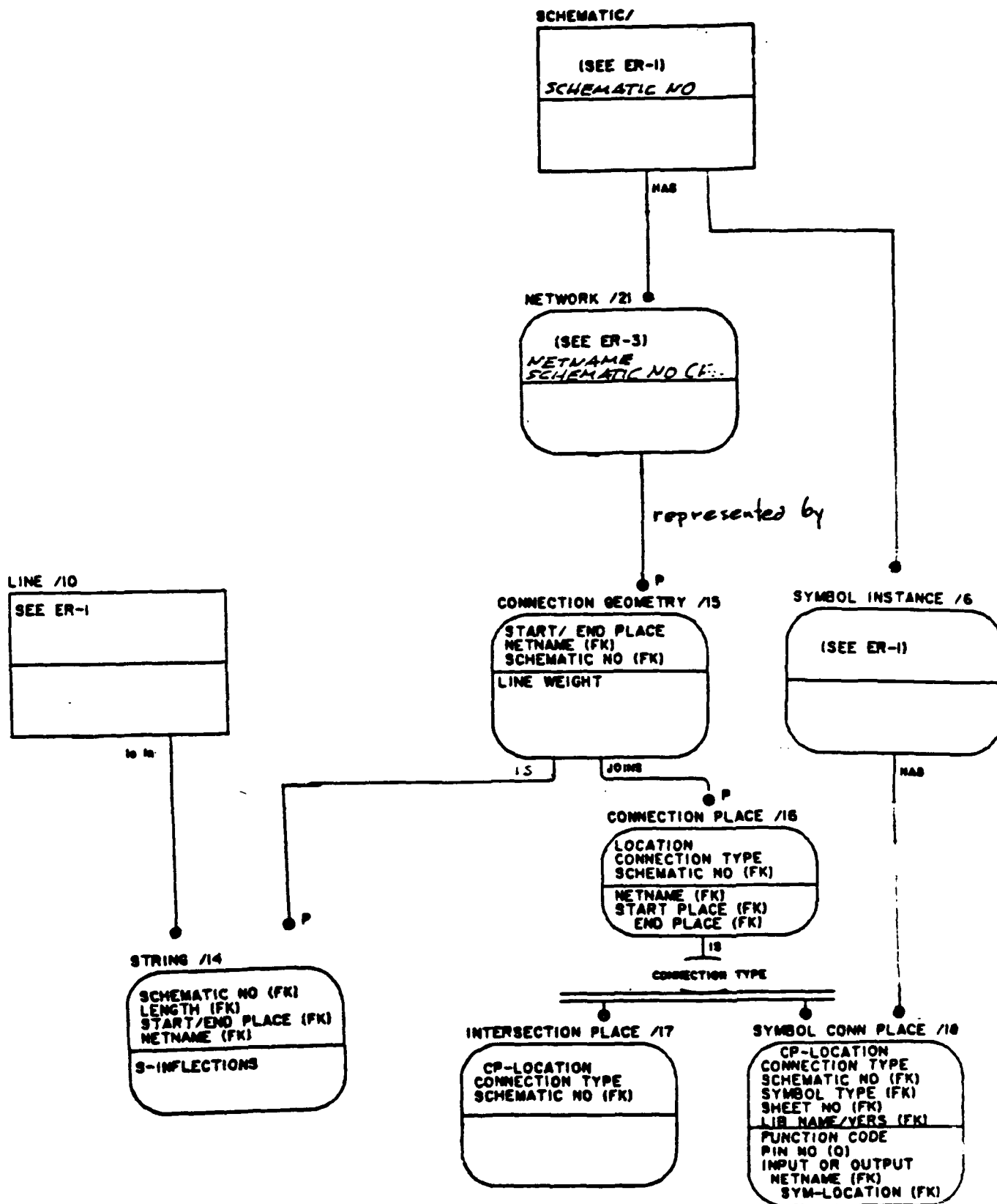SYMBOL INSTANCE: The occurrence of a symbol on a schematic.

Discussion Of Phase 1 Of The Methodology

We will describe the process of translating the Discipline Model (figure 4-1) to a Qualified model. Normally, this process starts by doing a literal translation from the Discipline Model. This will be a purely syntactical translation (figure 4-2) with no regard for the meaning of the objects in the diagram. The main purpose of this first-cut translation is to expose detail and verify any obvious key migration errors in the IDEF-1 diagrams. IDEF-1 places detail inside of the object boxes NIAM places detail outside of its object circles. We believe this detail-exposing process is critical to the qualification process. This mechanical translation was not possible in other application areas which did not use IDEF-1 as the modeling language.

Once the literal translation is available it is used as a reference by the logical layer person for learning the electrical schematic application. Our experience showed that we were able to ask questions to the application expert directly from the NIAM diagrams and the expert was able to answer the questions by examining only his IDEF1 diagram. There were times, however, when it was necessary for the logical layer person to have a good knowledge of IDEF-1.
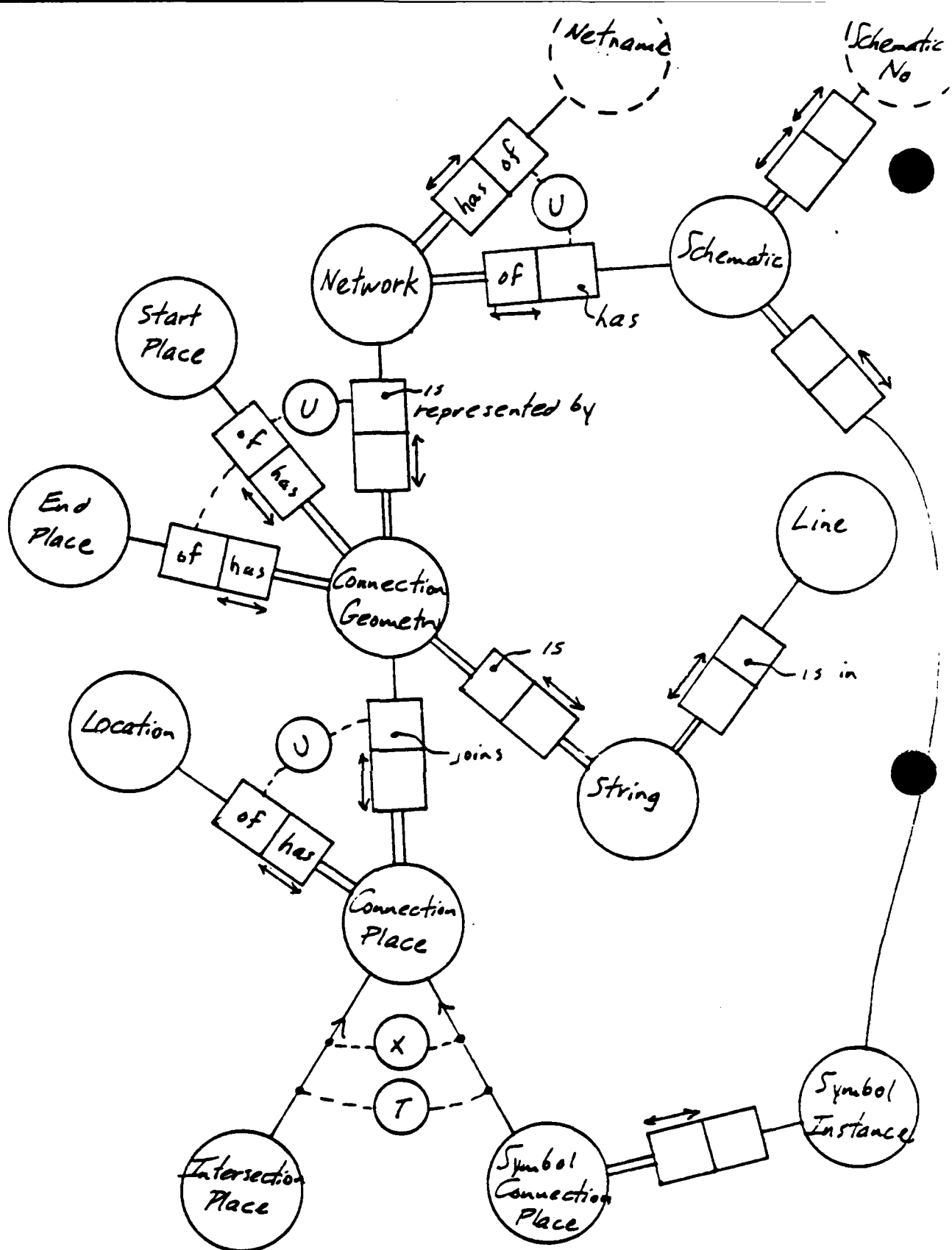
A specific technique used by the logical layer person in this specific example was to "picture" (figure 4-3) what the literal translation was saying. In some cases the original IDEF-1 diagrams lacked some precision in expressing concepts (translating literally to NIAM did not solve this problem). In those areas of imprecision, supporting text documentation was extremely helpful. It is always more difficult to formally model a concept than it is to casually express it in natural language. Once a picture is built, it is compared to the literal model. A constructive comment is offered at this point: most document packages contained few pictures of concepts. In most cases the logical layer person had to build his own pictures and validate with the application expert.

A new NIAM model (figure 4-4) is now constructed that agrees with the picture and the expert is asked to validate the new diagram. This diagram is the first version of the qualified model. This qualified diagram will go through five to seven

35

SCHEMATIC/

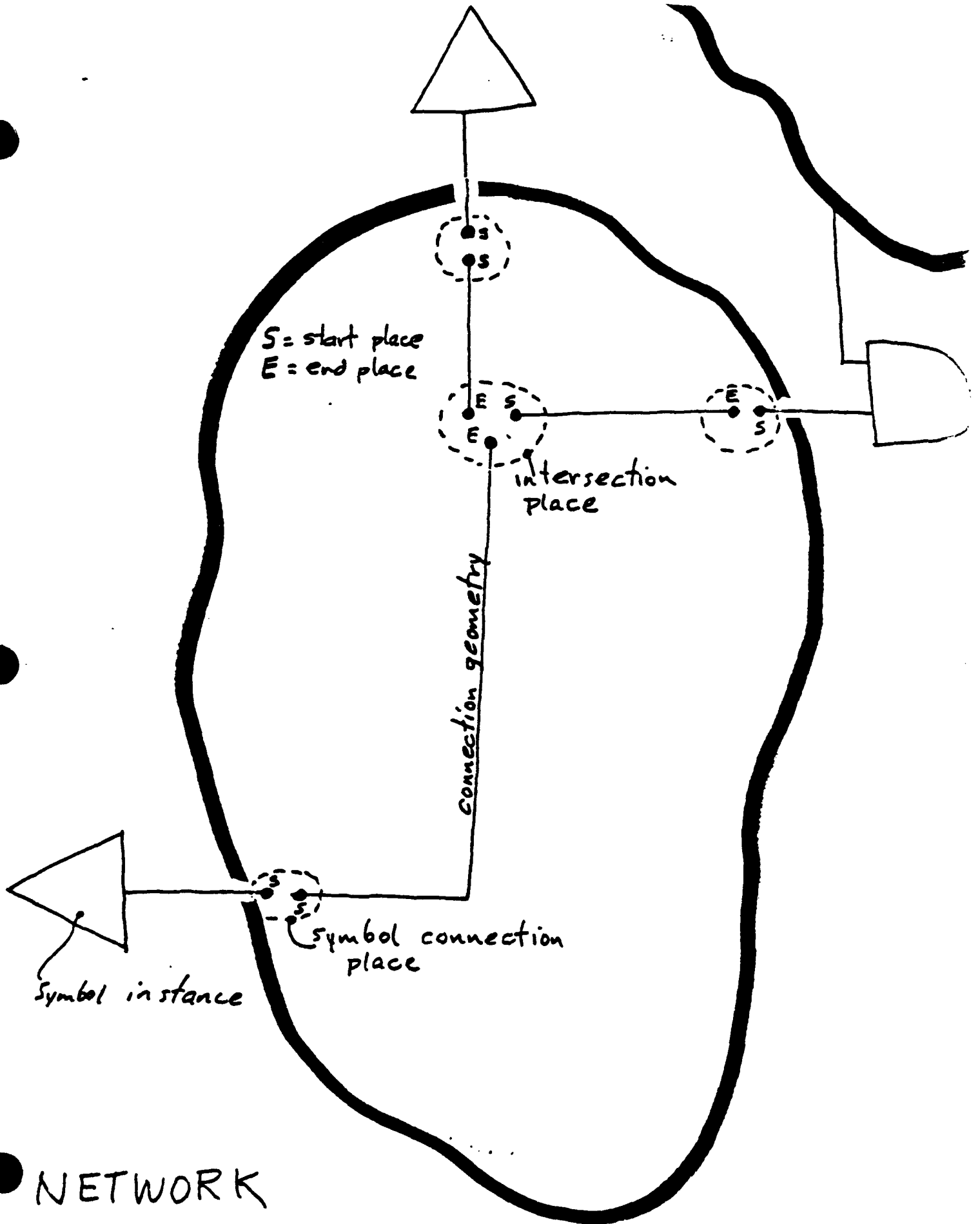(SEE ER-I)
SCHEMATIC NO

HAS

NETWORK /21

(SEE ER-3)
*NETNAME
SCHEMATIC NO CE...*

represented by

P

LINE /10

SEE ER-I

to in

CONNECTION GEOMETRY /15

START/ END PLACE
NETNAME (FK)
SCHEMATIC NO (FK)

LINE WEIGHT

IS   JOINS

P

SYMBOL INSTANCE /6

(SEE ER-I)

HAS

CONNECTION PLACE /16

LOCATION
CONNECTION TYPE
SCHEMATIC NO (FK)

NETNAME (FK)
START PLACE (FK)
END PLACE (FK)

IS

CONNECTION TYPE

STRING /14

SCHEMATIC NO (FK)
LENGTH (FK)
START/END PLACE (FK)
NETNAME (FK)

S-INFLECTIONS

INTERSECTION PLACE /17

CP-LOCATION
CONNECTION TYPE
SCHEMATIC NO (FK)

SYMBOL CONN PLACE /18

CP-LOCATION
CONNECTION TYPE
SCHEMATIC NO (FK)
SYMBOL TYPE (FK)
SHEET NO (FK)
LIB NAME/VERS (FK)

FUNCTION CODE
PIN NO (O)
INPUT OR OUTPUT
NETNAME (FK)
SYM-LOCATION (FK)

DISCIPLINE
MODEL

Figure 4-1

LITERAL
TRANSLATION

Figure 4-2

S = start place
E = end place

intersection place

connection geometry

symbol connection place

Symbol instance

NETWORK

IMPLIED CONNECTIVITY

Figure 4-3
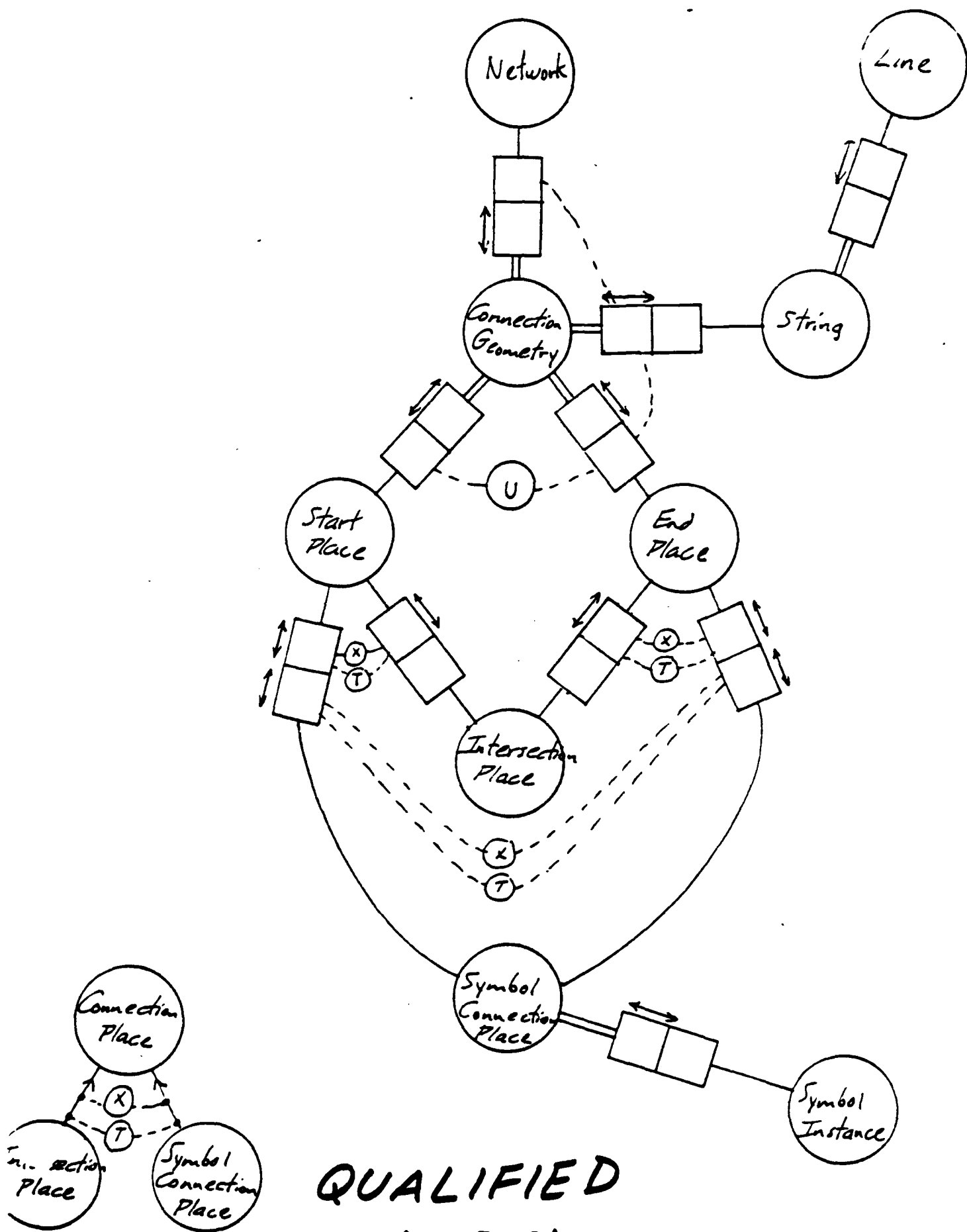
QUALIFIED
MODEL

iterations before complete agreement between application and logical layer is achieved.

In the case of the electrical model, some changes were made to the literally-translated model. It was felt by the logical layer that the original diagram did not precisely express the content of the picture in figure 4-3. The constraint capability of NIAM was used to add precision mostly through the total-role and mutual-exclusion role constraints. (Refer to NIAM primer in Appendix D2 for explaination of total-role and mutual-exclusion role constraints.)

The Literal translation basically said that Intersection Places and Symbol Connection Places are both kinds of Connection Places. But the association between Intersection Places and Start/End Place and between Symbol Connection Places and Start/End Place was not clear. The picture in the logical layer person's mind is shown in figure 4-3. The qualified model explicitly shows the above relationships and adds additional constraints. It cannot be overstressed how important this picturing process is as a lot of the common knowledge that application experts share at application working sessions is not available to the logical layer person.

4.1.3 Review Of Phase 2 Of The Methodology: Global Conceptualization and Integration

By the time we were ready to proceed with this phase, the geometry, presentation, and topology models had been completed. So the basic technique was to find a way to replace specific objects in the qualified model with generic objects from the resource models (geometry, presentation, topology). The notion of Place already existed in the geometry schema so it was used. Place appears only once in the global model instead of four times as in the qualified model. The four occurrences of place were replaced by a single occurrence of Place and three new roles. These roles are expressed in the following four sentences derived from the global model:

    Place2 is location of Symbol Connection.
    Place2 is location of Intersection.
    Place2 is end of curve segment.
    Place2 is start of curve segment.

The edge-vertex portion of the topology model is used to model connectivity explicitly. In the qualified model, there is no mechanism for explicitly showing connectivity within the model unless it is 1) implicitly assumed through symbolic sharing by Connection Geometry of a common geometric Place or 2) a rule that states "any start or end places that are located within delta x of each other are to be considered connected". This is the rule

36

the human would use in looking at a schematic. In most cases it is possible to visually determine that the start and end places actually touch and are therefore considered to be connected. The computer has no visual image of touching. The main purpose of the global model is to give the computer explicit "touching" information through the Geometry-topology associativities Edge-curve and Vertex-place A picture of explicit connectivity is shown in figure 4-5.

The fundamental test that must be run against the new global schema (figure 4-6) is: can a query be run against the new schema to create a net list? Casual examination of the global schema should confirm this. We do not have any thorough way of determining formally whether or not the qualified schema and global schema are equivalent.

As stated in the methodology description, integration is the act of reconciling the qualified model with the global model. We will attempt to do that in English as we have no formal mapping language. The general form of each reconciliation statement will be "X in qualified model is replaced by Y in global model." For example:

> The sentence "Connection Geometry has Start Place" in qualified model is replaced by the sentence "Edge-curve has connection line defined by Composite Segment2 that is composed of Curve Segment2 with start place defined by Place2" in the global model. We ask ourselves "Is this a valid sentence replacement?". If it is we have a successful instance of reconciliation.

It should be mentioned here that new connectivity information has been added in translation to the Global model. A critical issue here is whether or not to allow additions to be made to the qualified schema when converting it to global form as this confuses the process of reconciliation. Realistically this will probably happen as the logical layer person perceives a complete way of expressing application semantics. Once this discovery is made, a decision must be made to go back and make the qualified model complete. In the case of the electrical model, the qualified model was not upgraded because of schedule pressures.

Here are the most interesting things that one can discover from reviewing the global model (particularly figure 4-7 which shows the substitution by geometry, presentation, and topology):

1. A large portion of the qualified model was replaced by objects and structures from the resource models.
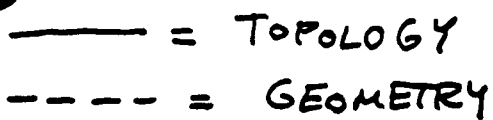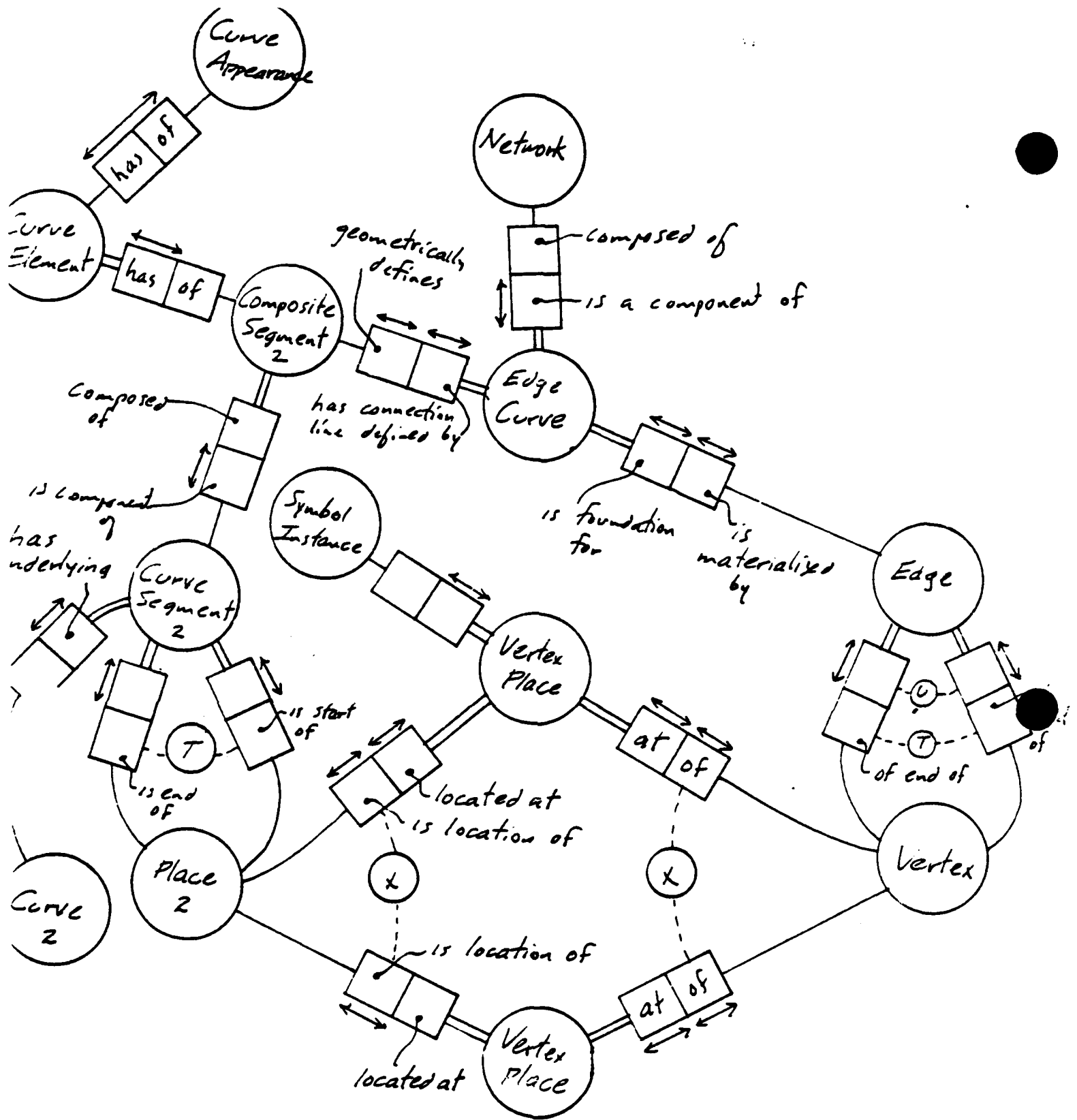
SEGMENT #2

intersection

SEGMENT #3

segment #1

Symbol connection

—— = TOPOLOGY   (— — —) = ASSOCIATIVITIES

- - - - - = GEOMETRY

Figure 4-5

EXPLICIT   CONNECTIVITY

GLOBAL MODEL

Figure 4-6

GLOBAL MODEL

Figure 4-7

2. Many objects in the qualified model have been replaced by associativities in the global model. These associativities act like "glue" to tie together fragments of the resource model.

3. Because the global model uses more general structures, it tends to have more objects.

4. The global model removes objects and adds new roles.

5. The resource object "Vertex-place" replaces both Intersection place and Symbol connection place.

6. The global schema is modular. Associativities tie the modules together.

4.1.4   Review Of Phase 3 Of The Methodology: Specification

Specification was broken into two basic steps: 1) grouping of the Global model and 2) translation of the groups into DSL. The grouping process was a joint effort between the builder of the global model and the DSL specifier. The groupings were based on the notion of defining relationships (figure 4-8). For example one group might be the association between Network and Edge-curve. Since the Global model says that a Network is composed of (defined by) many Edge-curves, this seems like a natural grouping.

Here are more specific examples of groups:

Edge-curve has connection line defined by Composite Segment2 and also is foundation for an Edge. (Note that the notion of defining relationship is somewhat artificial with associativities.)

Edge has start vertex. Edge has end vertex. (this is the normal Geometry-topology associativity)

Curve Segment2 has start Place2. Curve Segment2 has stop Place2. (this is the normal Geometry structure)

Vertex-place at Vertex and Vertex-place is located at Place2. (This is the normal Geometry-topology associativity structure - however, sufficient information has not been added to indicate that Vertex-place is playing the role of Symbol connection place. Note also that Vertex-place plays the role of being Intersection place. The orignal discipline model is almost unrecognizable in the Global model. It is important that we see an instance of a nearly "all-generic" Global model to see the importance of a mapping from Discipline model to Global model.
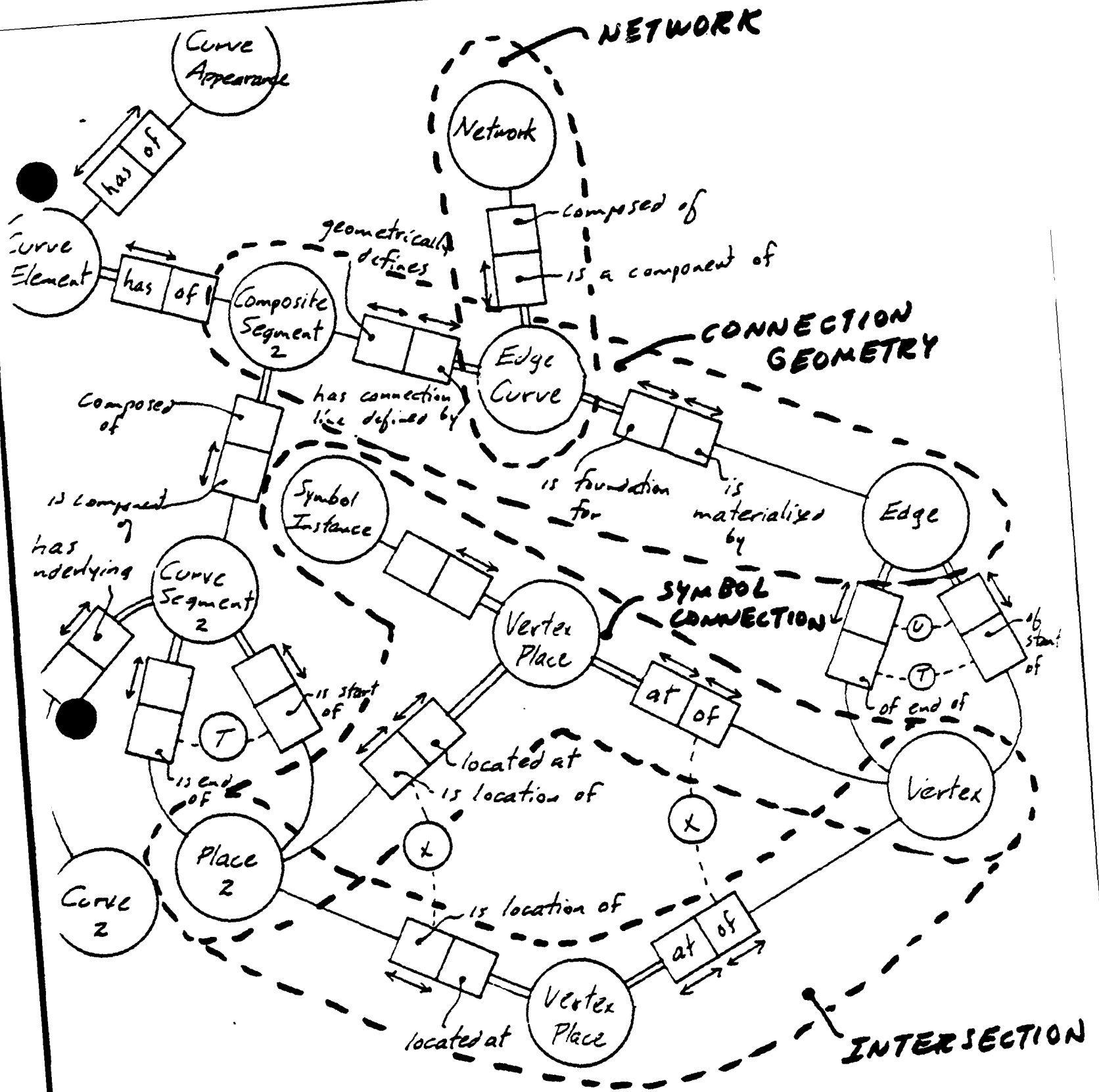
GLOBAL MODEL
GROUPING
Figure 4-8

We have attempted to group based on the loose notion of
"defining relationship".   We realize that some groupings may
be more difficult as the notion of definition is not as strong.
An associativity is something that has definition but the notion
of definition relationship is not so clear with abstract objects
such as associativites.   Does it make sense to say that Vertex-
place is defined by a Place2 and a vertex?   It does not sound
quite right.   There is some question as to whether the notion of
defining relationship can be clearly defined itself.   Grouping
will be an on-going problem that we must face if we are to
present the PDES in a concise easy-to-read form that abstracts
out detail.   Complexity is the real problem we are dealing with
and the need for abstraction techniques is critical.   Grouping is
an abstraction technique.   The Version 1.0 effort must come to an
agreement on a grouping rationale.

Here is a sample DSL of the groupings in figure 4-8.

```
ENTITY network;
ROLE
      component :LIST (1 to *) OF
                              REFER (edge_curve);
END;


ENTITY edge_curve;
ROLE
      geometry  :REFER (composite_segment_2);
      topology  :REFER (edge);
END;
```

4.2 Example 2: Flat Plate Mechanical Part Application Model

4.2.1 Introduction

The Logical Layer Group focused its attention on a specific application for its first attempt at exercising the methodology. The application chosen was the design of flat plate with holes. This application was purposely kept simple and based on the difficulty we had in modeling this application, it was a wise decision. As a result of this experience we are convinced that the principle of pushing specification applications through a methodology is a good test scheme. This kind of test is distinctly different from using a broad application area as a test. The knowledge area is very specific and limited, which reduces modeling time.

4.2.2 Review Of Phase 1 of the Methodology: Qualification

The complete documentation package received from the Flat Plate application group task group is contained in Appendix C3. This package clearly spelled out the scope and definition of the application. Our final model did not cover the entire area defined by the documentation package because of schedule pressures. Roughly, it covered the left half of the discipline model given in figure 4-9. For purposes of exposition, we offer a condensed version of the document package.

Scope of the Model

For the purposes of this initial document, flat plates are considered to have the following characteristics:

- Plate has two major (large) faces, a top and a bottom

- Uniform thickness; i.e., top and bottom faces are parallel.

- The top and bottom faces are not required to have congruent dimensions (this requirement was changed later to require congruent dimensions in order to simplify the model).

- The thickness is small relative to the top and bottom face dimensions.

- For any traverse from top to bottom face, along the intersection of an arbitrary plane perpendicular to the top and bottom faces, only one "side face" will be encountered.

40

Flat Plate Model Overview
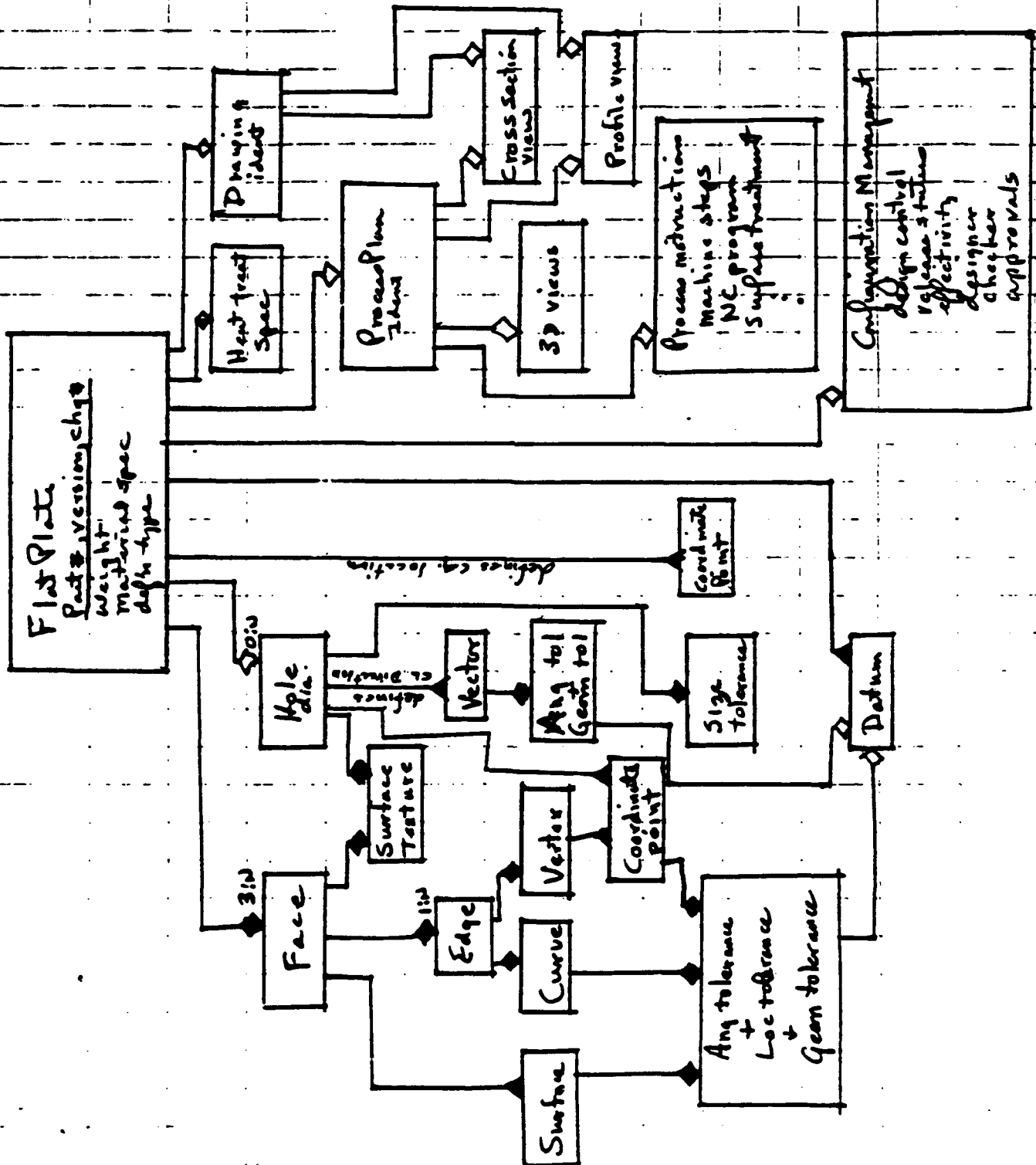
JS DePaum — 17 July 85

Figure 4-1

- Only round, through holes of uniform diameter are allowed. (This requirement was later relaxed as it was just as easy to allow for holes of arbitrary shape and depth).

- Edge conditions, e.g. chamfer/radius specifications will be treated as attributes. If the chamfers/radii are large enough to require more detailed definition, they are outside the scope of this model. (Again, this requirement is somewhat relaxed by allowing the perimeter of the flat plate to be any closed planar shape with not self-intersections).

Definitions

Flat Plate

This is an entity that is the "top of the tree" in the definition of a single piece part. All the definitional information for the part is under this entity.

Face

Face is a topological entity. It is essentially a bounded surface. The boundaries are topological entities, edges, with their end points defined by vertices. This is the relatively standard B-REP notation.

Surface

The geometric (mathematical) entity that defines the shape of a face.

Datum

One of three planes that in principle define the coordinate system of the plate. The three planes are mutually perpendicular and intersect in a point. That point is the origin. The planes or some manifestation of them are referenced as datums in tolerances and other entities.

Angular + Location + Geometric tolerances

These tolerances are defined in the tolerance model. Their presence in this model is to indicate the relationships of tolerances to the flat plate part.

Edge

The topological entity that bounds a face. It is essentially a geometric curve, bounded by end points (vertices).

Curve

The geometric (mathematical) entity that defines the shape of an edge.

Vertex

The topological entity that bounds an edge. It is essentially a geometric point.

Coordinate Point

The geometric (mathematical) entity that defines a location in space.

Hole

A feature of the flat plate. It is defined by its diameter, centering vector, and a coordinate point contained on a surface of the plate. This feature entity may be required, in some applications, to be transformed into topology and geometry, e.g., an assortment of faces and edges, but for purposes of model definition, this definition is informationally complete within the scope of the model.

Surface texture

This is an entity that describes the allowable deviation from perfection (at the micro level) of a surface. The exact definition is given by ANSI and ISO standards.

Vector

A geometric (mathematical) entity that defines a direction in space.

Size tolerance

One of the tolerance entities that applies to the diameter of a hole feature.

Discussion of Phase 1 of the Methodology

The flat plate discipline model (left half) was completed in an extended version of IDEF-1 as shown in figure 4-9. The following objects were excluded from the discipline model: heat treat specifications, drawing identification, process plan

identification, cross section view, profile view, 3D view, process instructions and configuration management. The qualified model in figure 4-10 is a literal translation of the left hand side of the discipline model in figure 4-9.

This model caused us to become aware of an issue that may occur in follow on PDES efforts: What kinds of objects would we expect an application expert to place in a discipline diagram? Will they be high-level engineering objects like form features or will they be foundational objects like vertex, edge, curve?

The qualification process started by making a literal translation of the discipline model (left side) into NIAM. The main purpose of such a first-cut translation is to expose detail and verify any obvious key migration errors in the discipline IDEF-1 diagram. In the case of the flat plate model there was little attribute and key information inside the entity boxes. This literal translation was accepted as the qualified model with the following simplifying assumptions added:

1.   Top and bottom faces are required to be congruent perimeters.

2.   All holes are perpendicular to top of flat plate.

3.   Non-through holes are not allowed.

4.   Any arbitrary flat-plate outline is allowed.

5.   Any shape hole is allowed.

It was felt by the logical layer that this qualified model captured enough aspects of the application to test the methodology.

4.2.3   Review of Phase 2 of the Methodology: Global Connceptualization and Integration

Ideally, the global cognitive model (figure 4-11) would not be a reconceptualization of the qualified model. In order to link this model to the only resource model we had at the time (geometry), the qualified model was significantly expanded to include form features. We believed that in the initiation effort we could have this kind of license in order to test the methodology, but in follow-on efforts, the amount of new meaning that is added to any qualified model must be carefully controlled. The primary duty of the logical layer is to convert the qualified model into a global form. We think a basic problem in this part of the intiation effort was that we were struggling to model this application with the limited conceptual resources at hand (wireframe geometry only). Later, there was no attempt to update the original qualified model as a result of additional

43

FLAT PLATE

Part No
Part
Version
Version No
Weight
U
Change
Material Spec
Flat Plate
Change No
Def'n Type
Size Tol
Dia.
Face
Hole
Surface
Vector
defines C.L. direction
Surface Texture
Edge
Tolerance B
Ang + Geom
Curve
Vertex
Coordinate Point
Tolerance A
Ang + Loc + Geom
Datum
Flat Plate
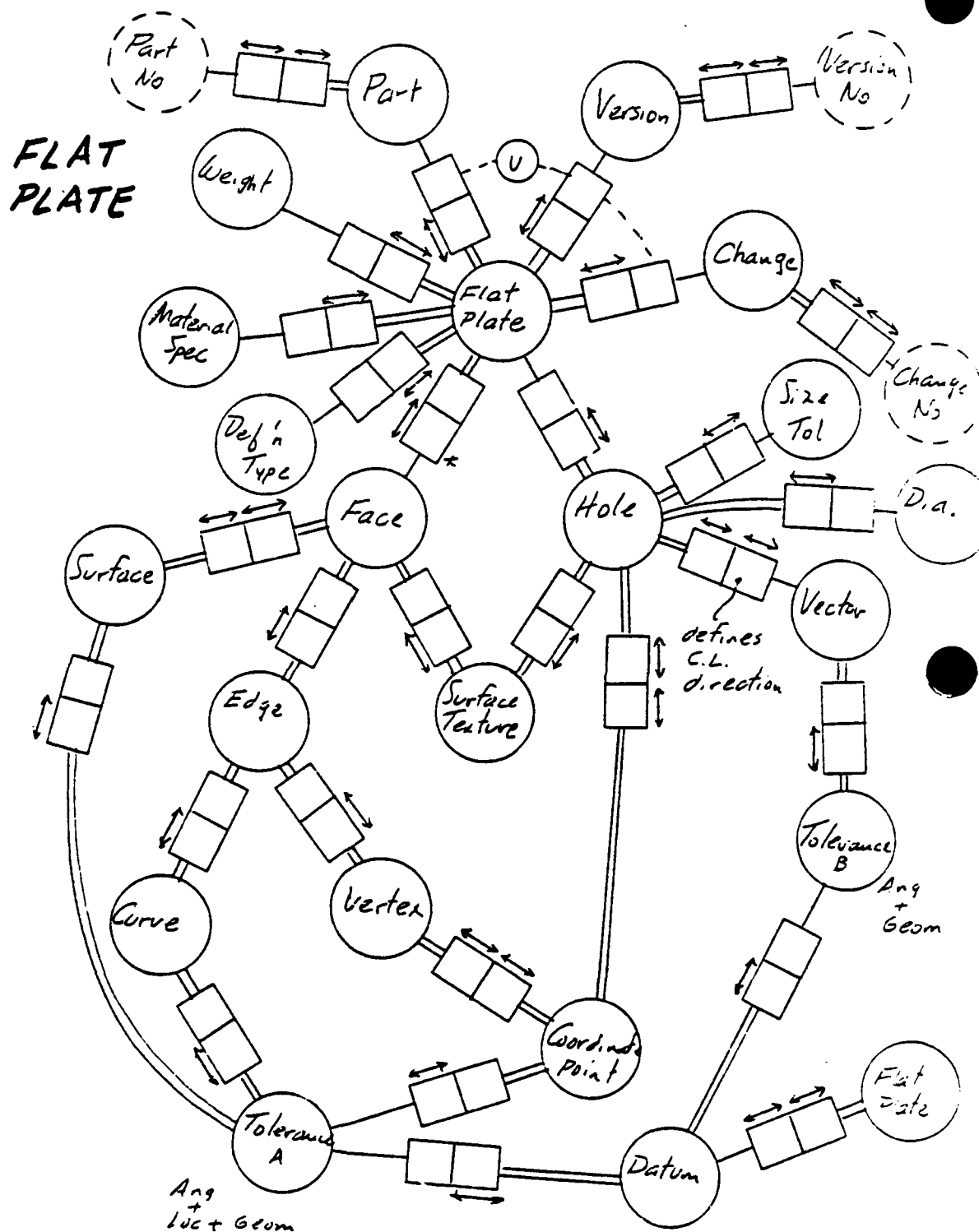
* 3:N cannot be expressed in the MIML subset of NIAM

LITERAL TRANSLATION AND QUALIFIED MODEL

Figure 4-10

GLOBAL MODEL

Figure 4-11

resource meaning (e.g., topology) added to the global model. Again, this should not be the case in future efforts.

The good that really came out of the flat plate effort was that we witnessed the evolution of a model. We think this is probably a typical real-world situation. Models don't just get built, they seem to evolve. Because we had only the geometry resource to start with, the flat plate model started out as a feature-based model with no explicit surfaces (we had no boundary model). Later in the project, a provisional tolerance model was created and became another resource for us. The global model then was incrementally modified to add explicit surfaces with surface attributes. This was done with little change to the form-feature part of the model.

A discussion of the changes to the qualified model follows:

1. The object "uniform thick profile" is added. This becomes the basic definitional object for the flat plate (instead of the explicit faces.)

2. The object "hole" is generalized to the object "uniform depression".

3. "Uniform depression" is in turn defined by "uniform thick profile" through the role "is negative body for". Note that the role of "uniform thick profile" for the flat plate body is "is positive body for".

4. The association object "flat plate uniform depression" is added to associate a particular uniform depression with a particular flat plate (and a composite model transform to position it).

5. The object "planar profile" is added to link "uniform thick profile" with the resource geometry at hand.

6. The object "face" in the qualified model is converted to "side(s)", "top", and "bottom". A relationship between "flat plate uniform depression" and "side" is added to indicate internal sides of the hole.

7. The object "Face-Surface" is added. This is a new resource object developed later in the initiation effort. Its purpose is to separate geometry and topology so that they can have independent existence. "Face surface" is among a class of resource objects called "Geometry-topology associativities" but originally were called "Primitive Shape Elements". We recognize that none of these are good names.

8. All surface references are now directed to the object "face
    surface". No surface property will be referred to either
    geometry by itself or topology by itself.

These changes, many of which involve additions or
generalizations, indicate that the logical layer actually stepped
beyond its legitimate bounds in this part of the methodology.
Phase 2 of the methodology should not normally result in
additions to the qualified model.

Actually, the phenomenon of the continuation of discipline
modeling on into the qualification and the Global
conceptualization and integration phases occurred across most
areas of the initiation work.

The global model presented here was a team effort involving
engineers, form feature experts, and information modelers. This
model is the fifth iteration.

4.2.4   Review of Phase 3 of the Methodology: Specification

Unfortunately, there was not agreement (between the people
responsible for the global model and the people responsible for
the specification model) on the content of the global model.
Figure 4-12 shows a possible grouping of the global model in
preparation for specification. Here is a sample of DSL that
corresponds to part of that grouping:

```
ENTITY flat_plate;
ROLE
     pos_bod    :REFER (uniform_thick_profile);
     holes      :LIST (1 to *) OF
                     REFER (flat_plate_uniform_depression);
     top        :REFER top;
     bottom     :REFER bottom;
     side       :LIST (1 to *) OF REFER side;
END;


ENTITY uniform_thick_profile;
ROLE
     thickness :t_magnitude;
     out_line  :REFER Planar_profile;
END;
```

45

FLAT PLATE

GROUPED MODEL (PARTIAL)

Figure 4-12

```
ENTITY flat_plate_uniform_depression;
ROLE
     depression      :REFER uniform_depression;
     transform       :REFER composite_model_transform;
END;


ENTITY uniform_depression;
ROLE
     neg_body  :REFER uniform_thick_profile;
END;
```

4.3 Example 3: Tolerancing Application Model

4.3.1   Introduction

This was a very challenging application to model because of its bulk. We did, however, find a similar pattern among the various tolerances defined in ANSI Y14.5M-1982 which accelerated our efforts. This was a good application for global modeling as the tolerance application makes heavy reference to geometry and topology and associations between them.

4.3.2   Review of Phase 1 of the Methodology: Qualification

The documentation package received from the tolerancing application group is in Appendix C5 . For the purpose of exposition we offer here a condensed version of the document package.

Scope

This document provides the definition of functional content for tolerancing practices as specified by ANSI Y14.5M-1982 and ISO 1101 and 1660. These specifications are considered to be functionally identical for the purposes of this effort.

This information model is intended to completely define all tolerance information specified by ANSI Y14.5M-1982 and the corresponding ISO specifications.

Excluded from Scope

Computing tolerances

Process tolerances

Pictorial representation of tolerances

Dimensioning practices

Non-mechanical part tolerances (e.g. electrical component values)

Assumptions

Product models are assumed to be 3D wireframe with surfaces.

Models contain exact definitions of nominal product geometry.

Topology constructs, where used in this model are included to satisfy the functional requirements of tolerancing. Since a

47

completely surfaced wireframe and a B-REP model are essentially equivalent, we have adopted the B-REP terminology of face, edge and vertex in our tolerance model.

General Definitions

Product: An item(s) which is(are) manufactured, or used in the manufacture of another item.

Model: A digital definition of a product.

Drawing: A pictorial representation of a product.

Dimension: A numerical value, implicit in the model geometry, which is a measure of the product.

Summary Of Entity Types

100 Geometry
unit vector, point vector, vector, curve, coordinate, point

200 Topology
vertex, edge, face

300 Feature
datum, conditioned datum, feature of size, form feature, face, edge, vertex

400 Tolerance
Coordinate tolerance: angle tolerance, location tolerance, size tolerance

Geometric tolerance: angularity, circular runout, circularity, concentricity, cylindricity, flatness, parallelism, perpendicularity, position, profile of a line, profile of a surface, straightness, total runout

Discipline Model Examples

Location Tolerance Entity (403): Allowable deviation of a measure of a feature from its design nominal position relative to a specific base location along a specified path (figure 4-13).

```
Start_entity     :Location_origin (503)
Origin           :Optional Location_path (504)
Path             :Array (1 to maxint) of
                     Location_toleranced_entity (502)
Basic            :Boolean
Plus-tol         :Real
Minus_tol        :Real
End-entity
```

MAR 26 1986

COORD-TOL 401

BASIC
PLUS-TOL
MINUS-TOL

LOCATION 403

COORD-TOL 401

USES A
AXIS FOR
DIMENSION

DIMENSION
MEASURED ALONG

LOC-PATH 504

LOC-ORIGIN 503

TOLERANCES

1M

LOC-TOL-ENTS 502
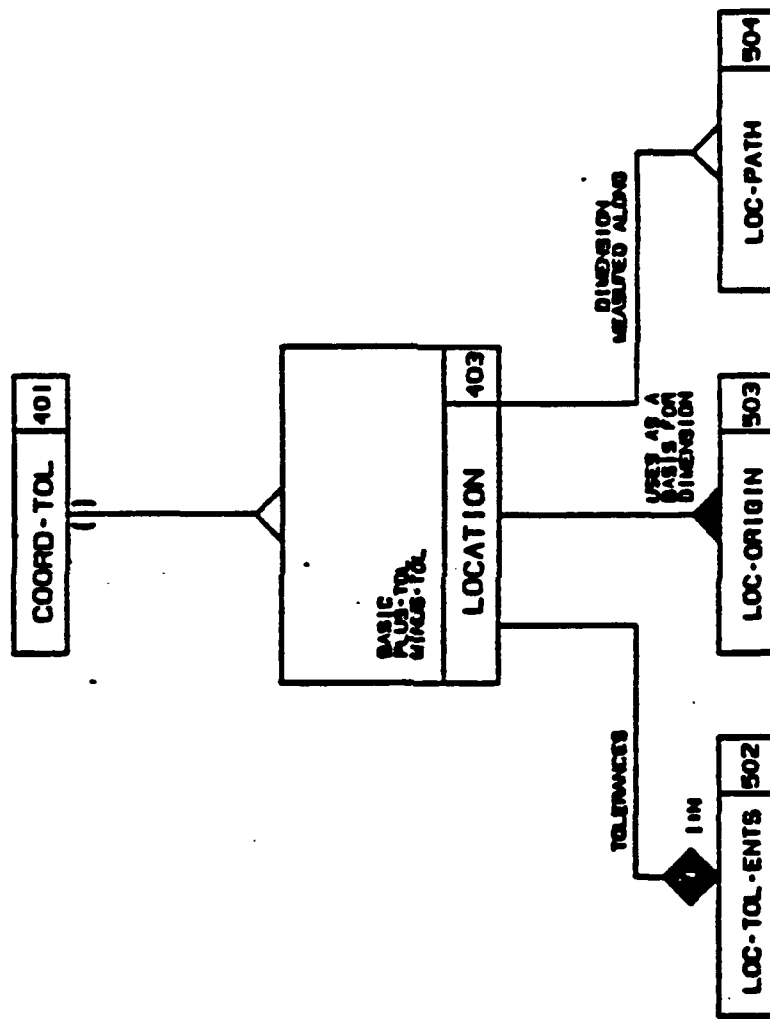
L0-403-LOCATION

Figure 4-13

Attribute descriptions:

Origin: An entity that serves as the base or origin of a calculated dimension. It is the "from" entity of the directed dimension.

Path: A curve along which (or in the direction of which) the dimension is measured.

Toleranced_entity: An array of entities to which the tolerance applies.

Basic: A boolean (true/false) flag that indicates whether the entity is used as a BASIC dimension.

Plus_tol: The absolute value of the tolerance that is added to the nominal dimension value to establish the maximum allowable deviation of the toleranced entity from the nominal.

Minus_tol: The absolute value of the tolerance that is subtracted from the nominal dimension value to establish the minimum allowable deviation of the toleranced entity from the nominal.

Location Toleranced Entity (502): Location toleranced entity is a location coordinate and a member from the class of edge (202), face (203), location tolerance qualified form feature (533), feature of size (303), vertex (201). See figure 4-14.

```
    Start_entity
    Toleranced_entity    :Face, edge, vertex, feature_of_
                             size, or location_tolerance_
                             qualified_form_feature
    Toleranced_location  :Coordinate (107)
    End_entity;
```

Attribute Descriptions:

Toleranced_entity: An entity to which the tolerance applies.

Toleranced_location: A coordinate which specifies the reference location on the toleranced entity. The coordinate must lie on the toleranced entity.

Location Origin Entity (503):
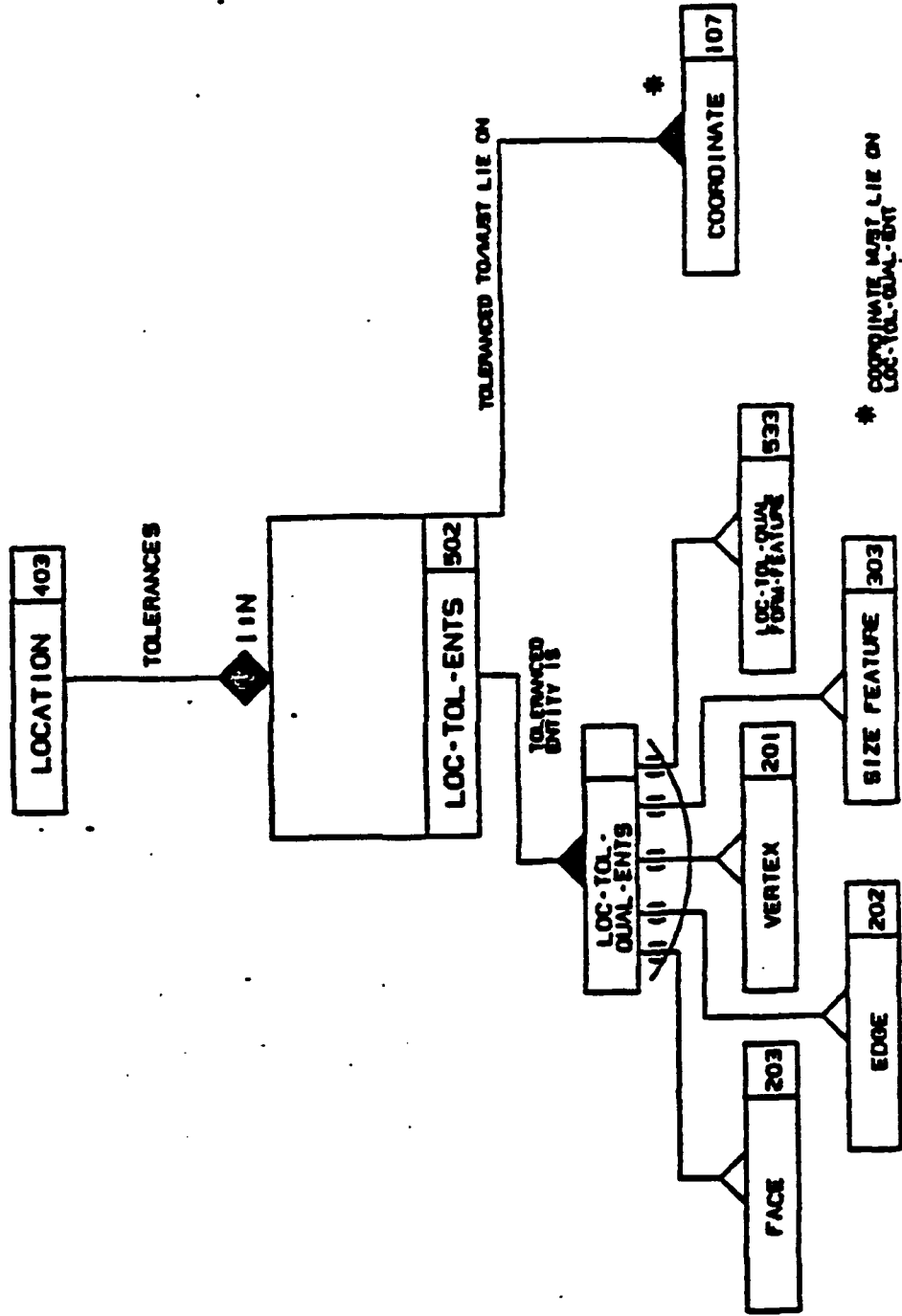
L7-502-LOCATION TOLERANCED ENTITIES

Figure 4-14

Location origin is an origin coordinate and a member from the class edge (202), face (203), datum (301), vertex (201). See figure 4-15.

```
Start_entity
Origin_entity        :Face, edge, vertex or datum
Origin_location      :Coordinate (107)
End_entity;
```

Attribute Descriptions:

Origin_entity: An entity that serves as the base of the calculated dimension. It is the "from" entity of the directed dimension.

Origin_location: A coordinate which specifies the base position. The coordinate must lie on the origin entity.

Location Path Entity (504): A location path is a class of entities. The class is used to specify the path along which a dimension measure is calculated. A curve instance must contain but necessarily end with the toleranced coordinate point(s). See figure 4-16.

```
Class of  :unit_vector (101)
           curve (105)
End_class;
```

Discussion of Phase 1 of the Methodology

From the vantage point of the logical layer this was a very complete and precise document that was at first overwhelming in its content. This document contained both IDEF-1 (extended) diagrams and a Pascal-like (similar to DSL) language of the tolerance model. We found that the IDEF-1 diagrams served as a guide to the entire model, whereas the Pascal-like specification seemed to be the focus of the document's precision.

Our liaison sessions were very intense. Two tolerance application persons acted as liaison in a visit to the logical layer liaison session. NIAM was used as the primary communication tool between tolerance liaison and logical layer, so in affect the translation was being done "live".

The qualified model covers almost all of the content of the document. Geometry and topology were modeled even though these models existed externally as a resource models. It was necessary to get the application view of geometry and topology in order to determine that the resource models were satisfactory.

We feel that, for the most part, this instance of qualification was indeed a translation from the application
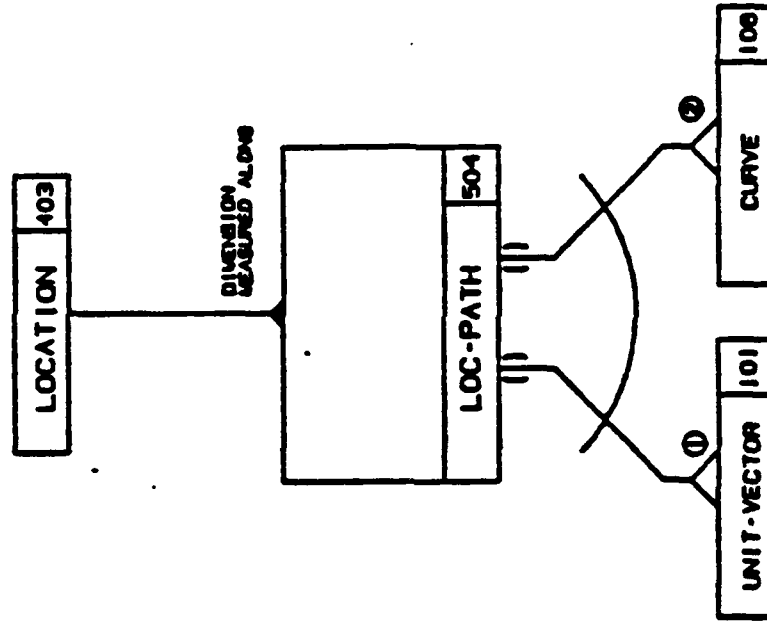
50
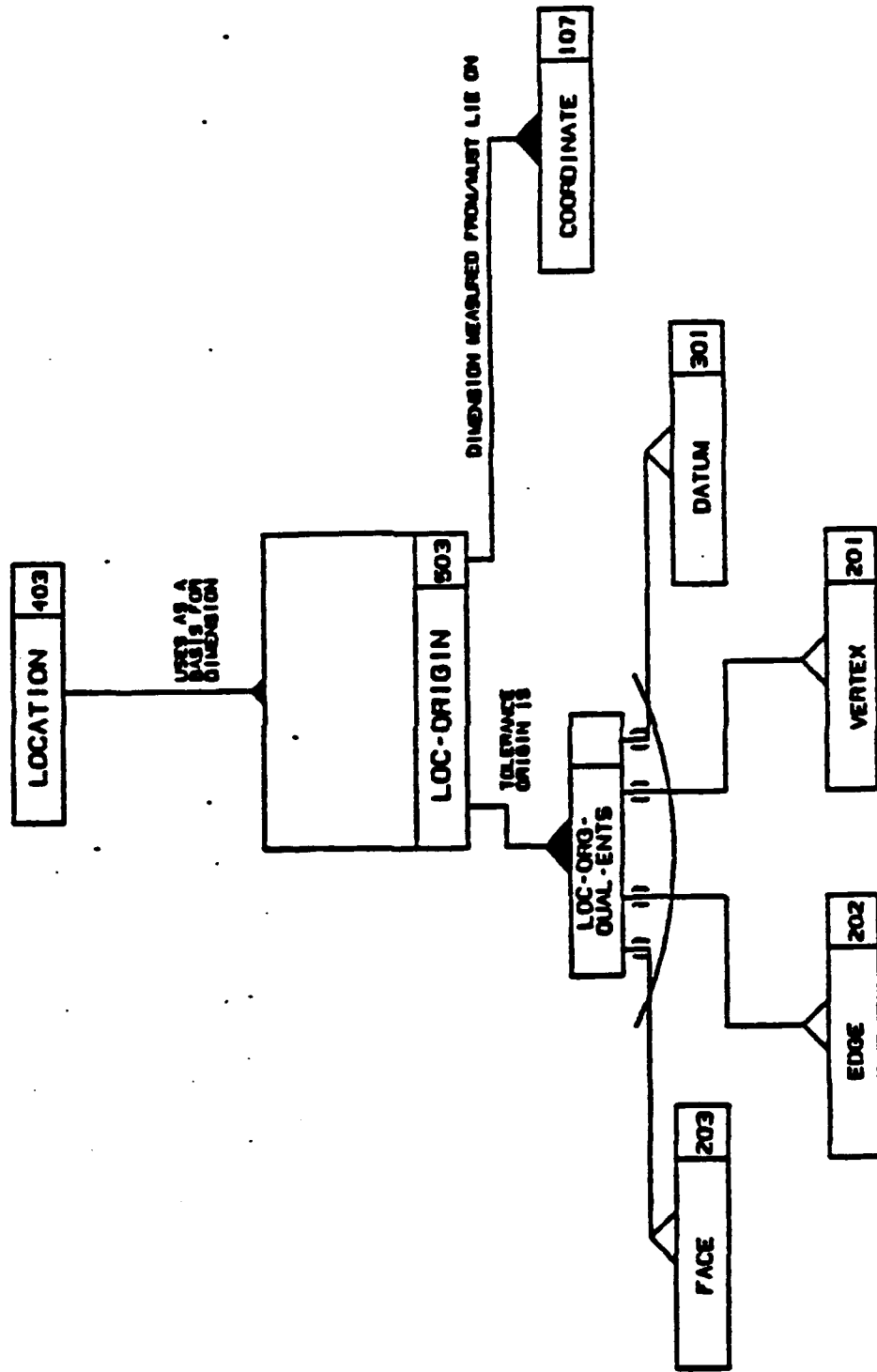
MAR 26 1986

L8-503-LOCATION ORIGIN

Figure 4-15

LOCATION | 403

DIMENSION MEASURED ALONG

LOC-PATH | 504

CURVE | 106

UNIT-VECTOR | 101

L9-504-LOCATION PATH

Figure 4-16

language and not merely a reconceptualization of the application area. There were a few times when the logical layer referred directly to the ANSI Y14.5M document for resolution of meaning.

Figure 4-17 is the qualified NIAM model corresponding to the above Location Tolerance (403). The translation from the Pascal_like syntax to NIAM was not mechanical. The two liaison persons participated heavily in this translation and in fact learned to read NIAM diagrams in order to perform their function. This is an ideal situation but one we do not expect to encounter in all application areas. We will probably be fortunate to have application experts fluent in one modeling language. However, we could not have achieved the translation in the time permitted by the schedules without the application experts having a reading knowledge of NIAM.

4.3.3 Review of Phase 2 of the Methodology: Global Conceptualization and Integration

The process of building the global model (figure 4-18) was a straight forward process of identifying objects in the qualified model that had corresponding objects in the resource models. As a result of building this global model, a new resource model was created. This new resource model is called "Primitive shape elements" (figure 4-19). This new model was created in order to keep geometry and topology as independent resource models. This model is a set of associativities between geometry and topology. Often models will refer to topology and then topology will refer to geometry (or does geometry refer to topology?). The "Primitive shape elements" model puts geometry on the same level as topology (one is not subordinate to the other). The principal objects in this model are: face-surface, edge-curve, and vertex-point. This model probably should be renamed as "primitive shape elements" is probably already used in many other contexts. We considered calling it GTA for geometry-topology associativity.

The following is a summary of substitutions that were made to the qualified model to guild the global model:

1. Face is replaced by Face-surface. (Primitive shape elements).
2. Edge is replaced by Edge-curve. (Primitive shape elements).
3. Vertex is replaced by Vertex-point. (Primitive shape elements).
4. Coordinate is replaced by Place (Geometry).
5. Unit vector replaced by Direction (Geometry).
6. Topology replaced by Primitive shape elements.
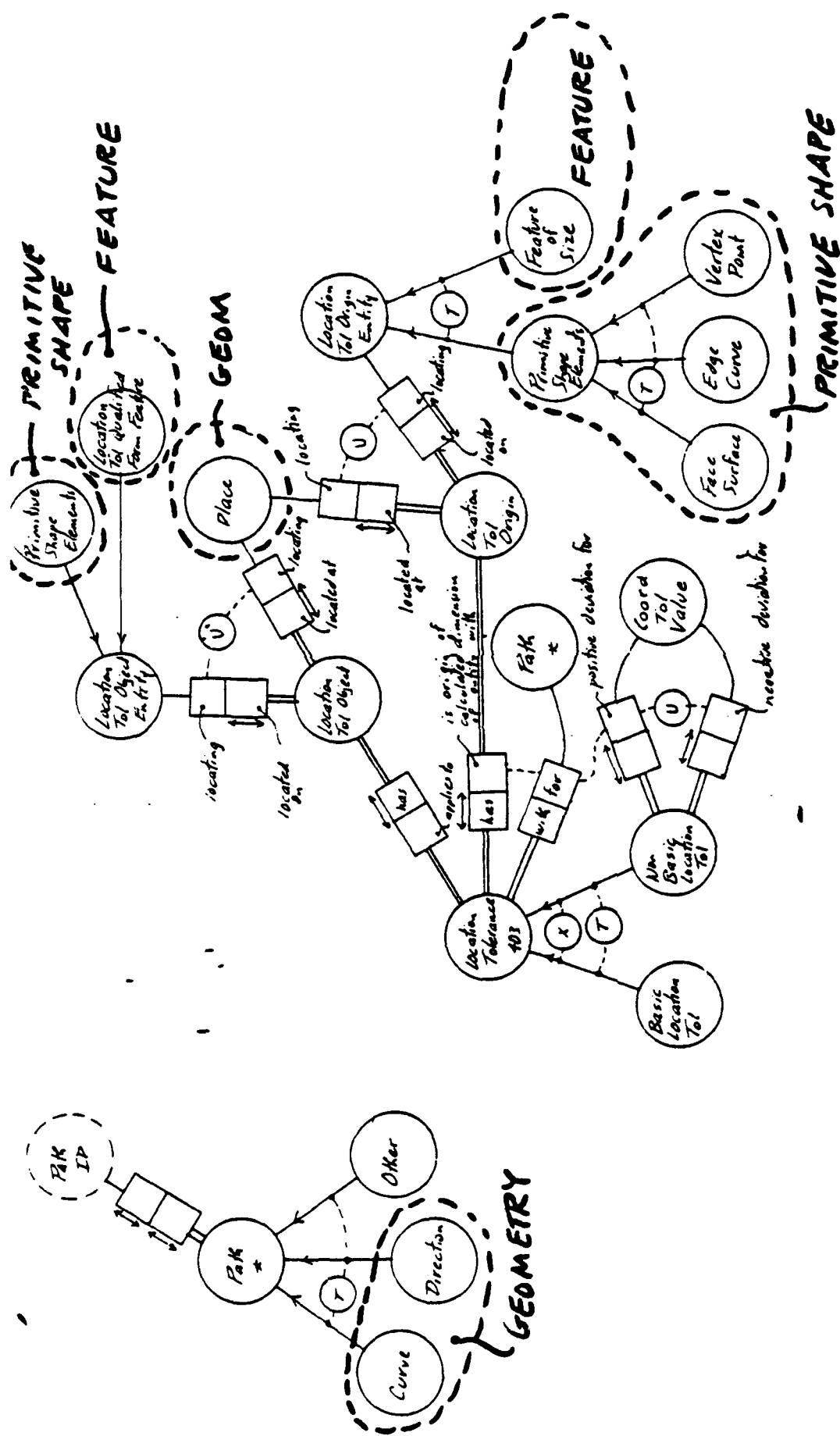
51
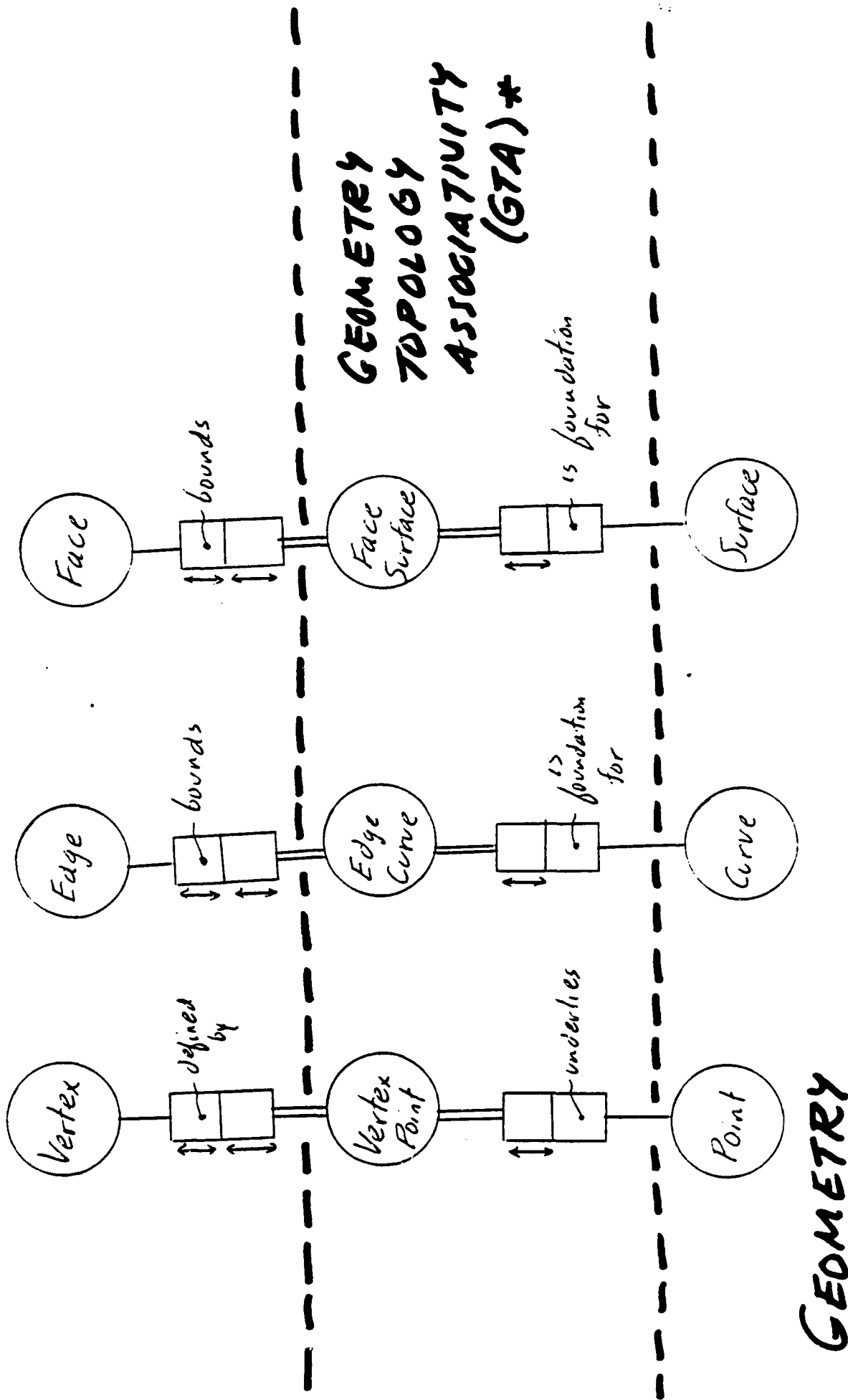
QUALIFIED MODEL

403 LOCATION TOLERANCE

Figure 4-17

GLOBAL MODEL

403 LOCATION TOLERANCE

Figure 4-18

TOPOLOGY

Vertex — defined by — Vertex Point — underlies — Point

Edge — bounds — Edge Curve — is foundation for — Curve

Face — bounds — Face Surface — is foundation for — Surface

GEOMETRY TOPOLOGY ASSOCIATIVITY (GTA)*

GEOMETRY

* Also referenced to as "Primitive Shape"

Figure 4-19

A new resource model called "Feature" was considered but was not followed because it represented a new arena of modeling problems. We do think, however, that there should someday exist a feature resource model.

4.4   Example 4:   Finite Element Modeling Application Model

4.4.1   Introduction

The FEM discipline model represents a significant team effort in acquiring the conceptually modeled FEM knowledge. The FEM team had as an objective the creation of a single model broad enough to accommodate any major FEM application. This model was particularly useful to the initiation effort because it became stable very early in the project. This allowed the logical layer to really do its job of validation and global conceptualization. See Appendix C6 for complete model.

4.4.2   Review of Phase 1 of the Methodology: Qualification

The discipline model is shown in figure 4-20. The basic application documentation consists of a single IDEF-1 diagram and a definition of terms. Based on this documentation the logical layer was able to understand most of the application area with the exception of the FEM notion of connectivity which required additional liaison with the application experts. We were most fortunate that the FEM liaison person was employed by the same company as the principal logical layer worker. We were also fortunate that this model matured quite early so that the qualification process did not involve any extensive remodeling but rather concentrated on the validation process. While validating this model, extensive work was done in developing translation from IDEF-1 to NIAM. The qualification process involved the derivation of natural language sentences from the original IDEF-1 diagrams. These sentences resulted in adjustments to the model.

The model appeared to be too simple when it was first viewed by the logical layer. We were expecting a much larger model. What we had expected to see was an enumeration of many specific types of FEM elements. What we found was a very general model that would allow for any kind of FEM structure.

For the purposes of exposition, we offer here a condensed version of the FEM document package.
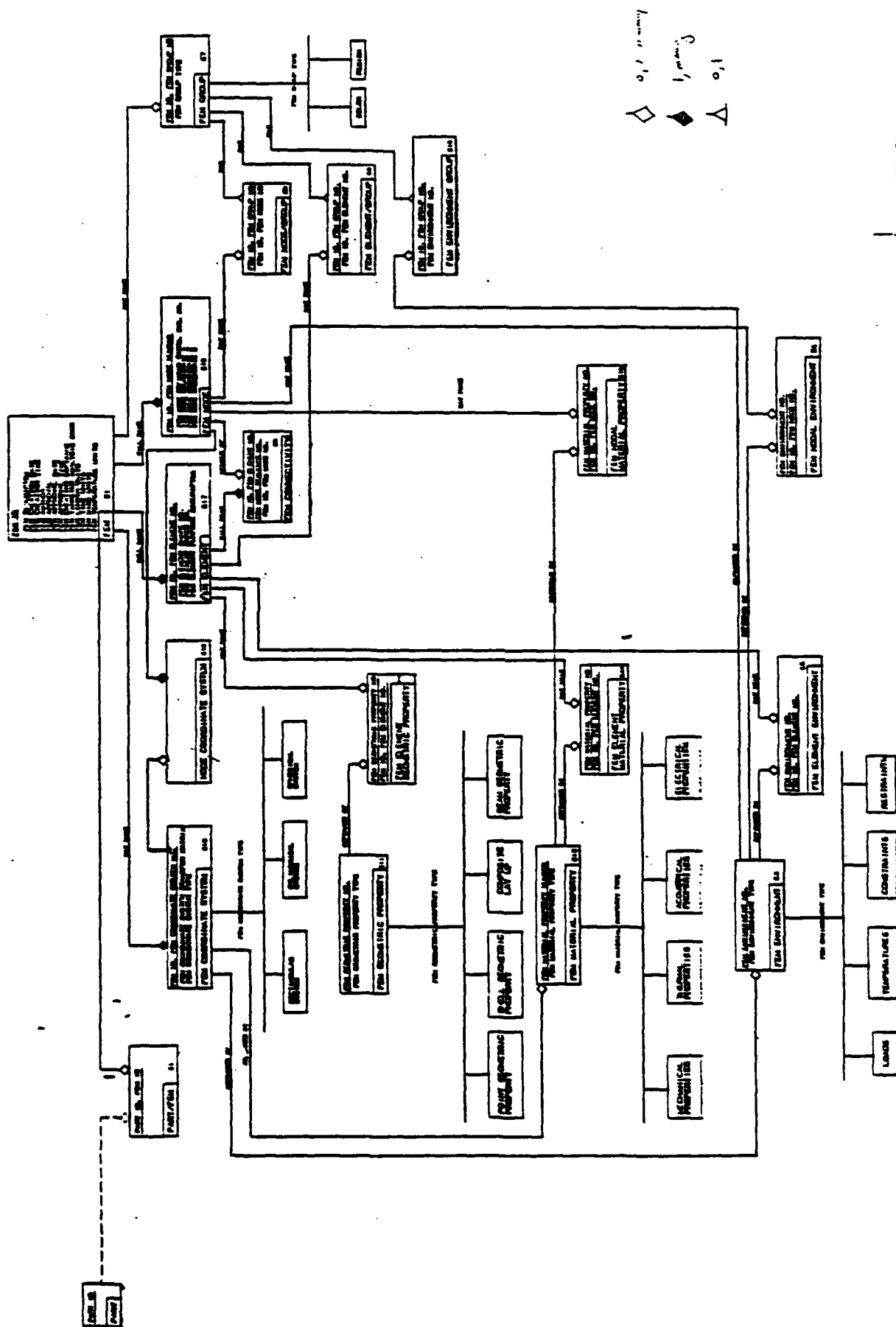
Scope

The model contains objects and relationships to accommodate all FEM elements types and all major FEM applications.

Definitions

Connectivity: the assignment of a node to an element. This refers to a single occurence of this cross reference (CR), not a list of all the multiple occurrences of this entity.

IDEF1 MODEL OF FINITE ELEMENT MODEL (FEM)

Figure 4-20

Coordination system: a frame of reference used to define the location of a finite element model or its components into 3D space.

Element: the basic model building block defining the relationship between its nodes.

Element/geometric property: assignment of a geometric property to an element. This refers to a single occurence of this cross reference (CR), not a list of all the multiple occurrences of this entity.

Element order: the mathematical definition for the allowable deformation of an element. An element with nodes only at corners is a linear or first order element, and may undergo only linear deformation. An element with an additional node on each edge between corner nodes is a parabolic or second order element. The order equals the number of non-corner nodes per element edge plus one. The element order and the element shape together imply an expected number of nodes to define the element. Missing nodes are caused by transition elements, and excess nodes imply face-located nodes.

Environment: any external or internal influence on a finite element model.

Environment/group cross reference: assignment of an environment to a group. This refers to a single occurence of this cross reference (CR), not a list of all the multiple occurrences of this entity.

Geometric property: values that may be used to describe the physical characteristics of an element. For example, shell element thickness, beam element area moment of inertia, etc.

Group: a collection of model nodes, model elements or FEM environments or any combination thereof.

Material Property: values that may be used to describe the constitutive nature of an element or a node.

Node: a location in a finite element model; used to define elements and environments.

Node/coordinate system cross reference: the assignment of a coordinate system to a model node. This refers to a single occurence of this cross reference, not a list of all the multiple occurrences of this entity.

Node/environment cross reference: the assignment of an environment to a node. This refers to a single occurence of this cross reference, not a list of all the multiple occurrences of this entity.

Node sequence number: the number which represents the position of a node in an ordered list of nodes which defines an element, (the connectivity list). A single instance of the connectivity CR entity will use this sequence number to cross reference a node with a particular location on a particular element.

Origin: the position in a coordinate system with all zero coordinate values.

Transformation matrix: the matrix used to specify the orientation and location of an entity in space, as well as the scale of the entity.

Discussion of Phase 1 of the Methodology

The qualified model is shown in figure 4-21. For the most part, the qualified model is a literal translation of the discipline model. The only part of the model that was changed was in the area of groups, classes of groups, and entities that are capable of being grouped. The ease of translation reflects the stability of the model. This model was modified less than any other discipline model in the initiation effort. As a result of the conversion to NIAM and a grouping of the NIAM diagram to normal form, a few key errors were detected.

Some of the application areas did not have a reading knowledge of NIAM and this became a hardship on the logical layer. It is nearly impossible to achieve qualification without at least one member of the application team (preferably the liaison person) being able to read and understand the NIAM diagrams. It was felt that the reading knowledge shown by the FEM group greatly accelerated the validation process.

4.4.3   Review of Phase 2 of the Methodology: Global Conceptualization and Integration

The global model is shown in figure 4-22. Only the area of FEM connectivity and FEM coordinate system were replaced by generic objects in converting the qualified model to a global model. In the discipline model the connectivity of nodes is defined at the element level by an ordered list. This array was replaced by the vertex-edge portion of the topology model. The intention was to use a single consistent method of modeling connectivity.

It was decided by the logical layer that the notion of associating multiple coordinate systems with a "Place" is common
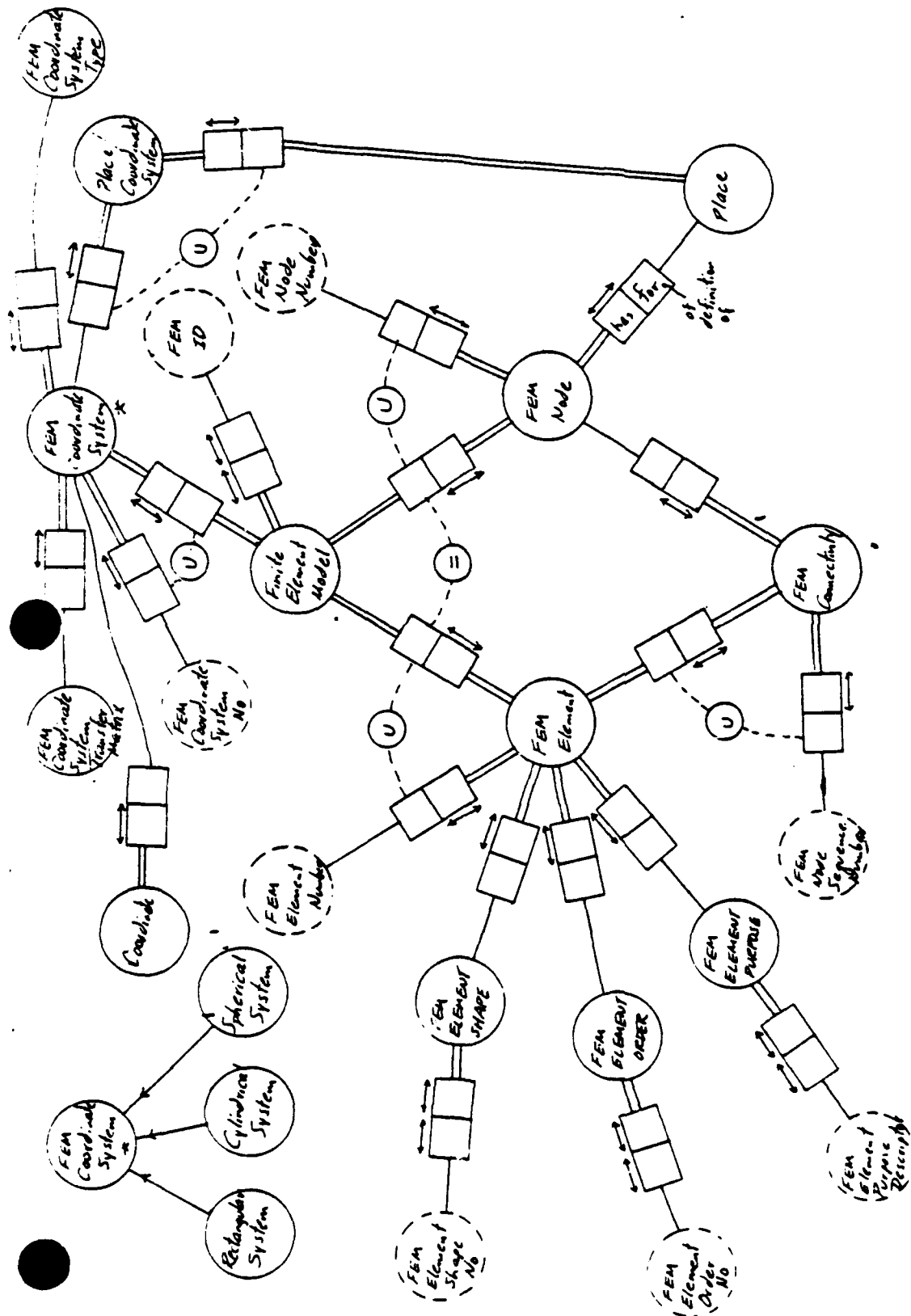
55

FEM
QUALIFIED

Figure 4-21

2-4-06

FEM INTEGRATED

FIGURE 4-22

Note: all element shape is buried in this object. Diagram shows no explicit element geometry. Global topology is determined from topology of fem elements and association of element vertices to fem node.
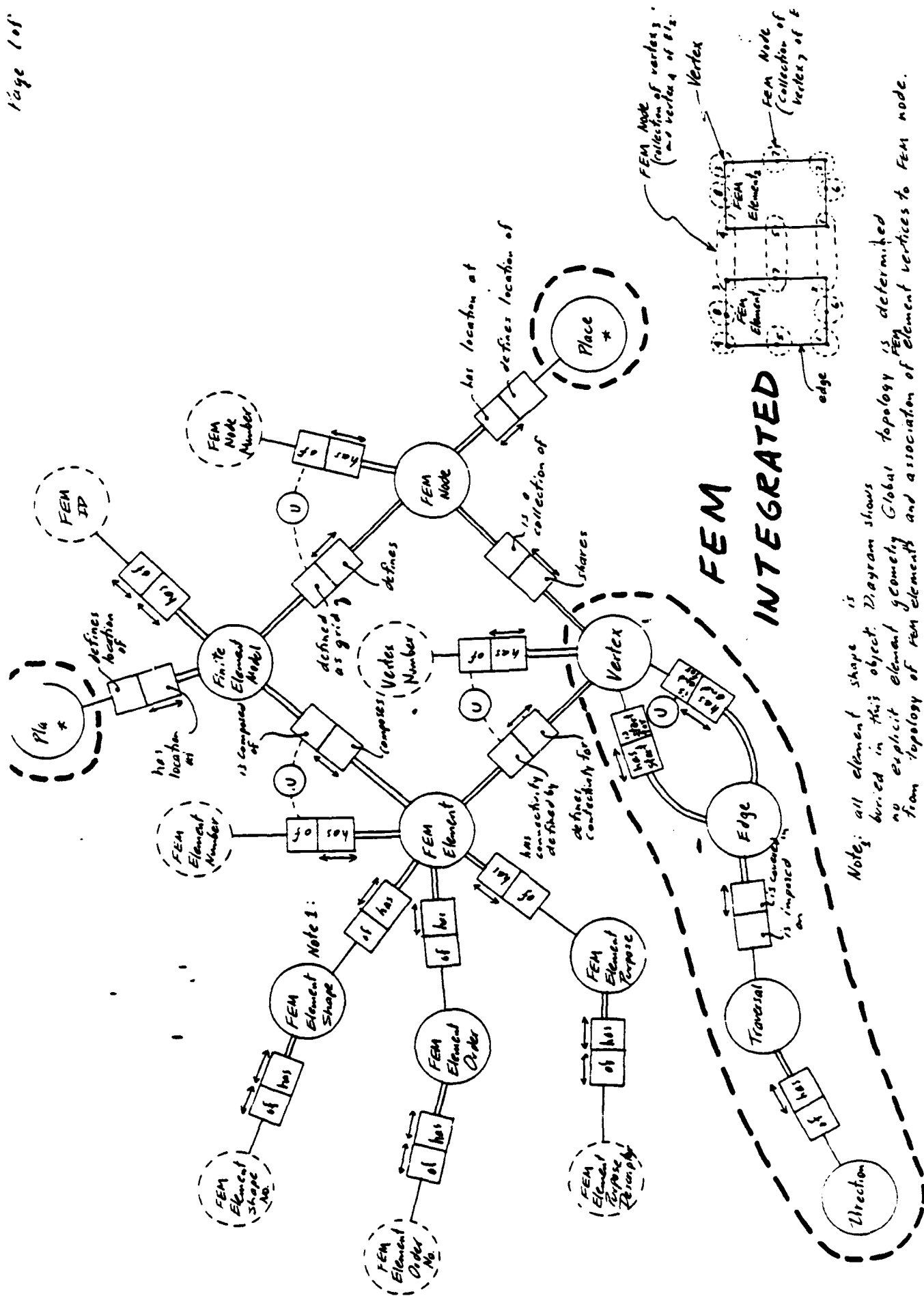
to many applications. In the global model of FEM there are no
longer any references to coordinate system. It is suggested that
the geometry resource model be changed to add the notion of
associating multiple coordinate systems with a place. This is a
good example of how resource models will evolve as more
applications are encountered.

4.5 Example 5: Wireframe Geometry Resource Model

Wireframe geometry entities were accepted as generic entities directly. Therefore, only Phase 1 of the methodology, qualification, exists as a task to be accomplished. See Appendix C1 for complete model.

We intend here to give a very high level overview of the content of the geometry model. Refer to figure 4-23 for a taxonomy of the objects in the geometry model. Notice that the basic breakdown is between wireframe geometric entities and wireframe auxiliary entities. The wireframe geometric segment is further broken down into Point and wireframe geometric Segment. The main auxiliary entities are Curve, Position, and Transformation Matrix. 2D and 3D geometry are shown as mutually exclusive subsets of geometry.

Notice also the key relationship between wireframe geometric Segment and wireframe auxiliary Curve which reads "wireframe geometric segment is a bounded, connected, oriented portion of a wireframe auxiliary curve." Here we see that the notions of boundedness, connectivity, and orientation separate pure geometry from auxiliary geometry. The auxiliary objects serve as unbounded "building blocks" for the geometric segments.

Although parameterization was addressed within the wireframe geometry task group, it was not modeled. It is merely indicated on the NIAM diagram that all auxiliary curves must have a parameterization. It has been questioned by some members of the logical layer whether parameterization is actually a conceptual notion. Considerable time and effort was spent attempting to define parameterization before deleting it from the model because we could not model it adequately.

The appearance of "Place" in the model warrants a comment. Place and Point both specify a single coordinate location in Euclidean space. The difference between the two lies in their intended use in the model. Point is used as model geometry. Place is used to designate location when what is being located in non-model geometry, such as the center of a circle.

We discovered constraints in geometry that cannot be modeled by the MIML subset of NIAM. However, by using more advanced constraint languages within NIAM (not contained in MIML) we were able to model the geometry constraints. There are also problems that occur by not being able to model local contexts and concepts such as instance and prototype, first class objects, and copied objects. From these examples, we see the feasibility that simple data base semantic modeling techniques, by themsleves may not be adequate for the PDES modeling effort.

57

WIRE FRAME ENTITY

12-17-85

Sheet 1

- We noticed while doing the FEM model that multiple coordinate systems were used. We recommend that the geometry model be updated to model the notion that a given place may be defined in multiple coordinate systems. We suspect that this may be a requirement for many applications.

More modeling work needs to be done on the spline as well. This topic was also addressed by the wireframe geometry task group.

4.6 Example 6: Presentation Resource Model

Presentation is the act of making product models visible and is not inherently part of product definition. However, presentation data is often transferred between dissimilar systems. Because presentation is not part of product definition, some felt it should not be part of our initiation modeling effort. See Appendix C2 for complete model.

We have chosen a compromise position by going ahead and modeling presentation but loosely coupling it to all other major classes of objects such as geometry and text. Presentation is not inherently a part of any other model and is related to other models only through associativities.

After considerable study on modern graphics techniques the logical layer came to the conclusion that in a modern graphics system there will exist two structures: 1) the product structure and 2) the presentation structure which will, to a great extent, mirror the product structure. Keeping these structures separate but related (presentation structure is derived from product structure) seems to be the key to modern CAD/CAM systems. What one sees in the PDES presentation model then is the NIAM version of the presentation structure developed by the presentation task group. Note that the only links between this model and the geometry and text models are the associativity objects "curve element", "surface element" and "text element". See figure 4-24.

The presentation model was the most difficult of all to build and consumed a month and a half of effort. The presentation task group documentation was voluminous and in places difficult to understand (mostly because of the logical layer workers lack of familiarity with modern graphics standards). Of the month and a half effort almost a month was spent in solid research on modern presentation standards and techniques. The entire PHIGS document was read as well as most of a popular college text book on graphics. Putting it bluntly, this model was a real "killer". The opinion of the logical layer worker was that he could not have made the model without the research.

The model went through approximately five iterations before it began to have a reasonable correspondence with the presentation task force model. Part of the difficulty was in interpreting the Pascal-like models in the presentation task force documentation. In many cases the modeling had to be done directly from English narrative.

figure 4-24 is an overview of the logical layer presentation model. It basically shows that a presentation model is the model of a picture that has sub-picture parts. Each picture part is composed of picture part elements which could in turn be another
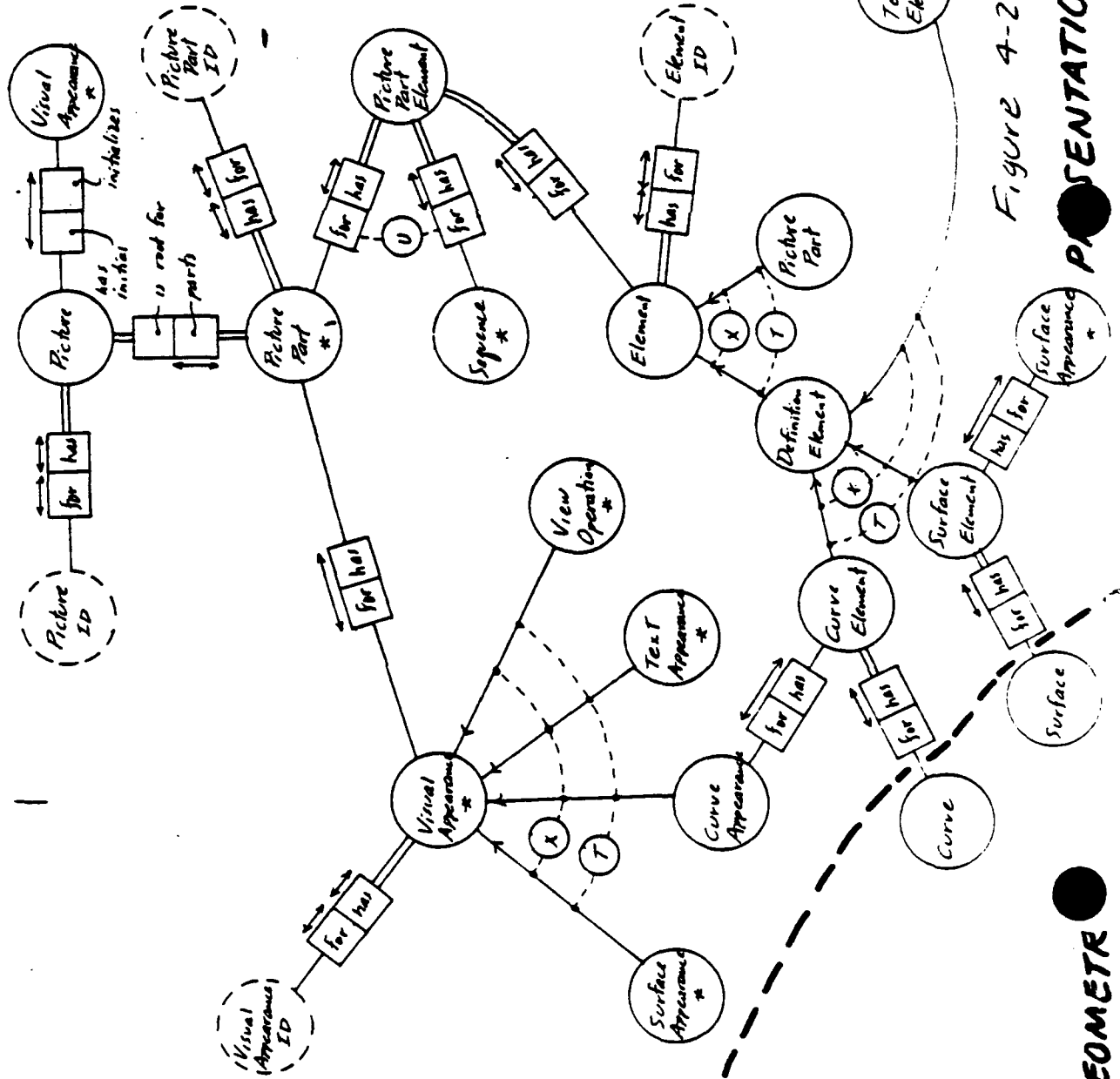
59

Figure 4-24

Debb, Vdscs,

PRESENTATION

TE:

GEOMETR

picture part or a specific association between a definitional element (geometry or text) and some number of visual appearance attributes. Note that it is not mandatory for the picture part "assembly structure" to mirror the product definition structure since they are only loosely associated at the elementary geometry and text element level.

The most important aspect of this model is that it has a structure of its own entirely apart from any definition of text or geometry. This is a significant departure from the IGES world.

4.7   Example 7: Topology Resource Model

The model in figure 4-25 is the temporary model referred to earlier in this report. It is basically the pure topology elements lifted from the IGES ESP boundary representation model. See Appendix C7 for complete model. Again, we mention that little global conceptualization could be accomplished without this provisional model. It is envisioned that an entire class of geometry-topology associativities will eventually emerge as future PDES modeling efforts are pursued. It was our desire that this model stand by itself with no references to geometry. The following sentences are taken from the NIAM model and highlight the major concepts:

1.   A vertex is either the start of and edge, end of an edge or a degenerate edge for a loop.

2.   A vertex may participate in the role of being both start and end of the edge.

3.   If a vertex is playing the role of being a degenerate edge for a loop it cannot simultaneously be playing the role of being either the start or end of an edge.

4.   A traversal may be imposed on an edge.

5.   An edge must have a start vertex and an end vertex.

6.   An edge is distinguished by its start and an end vertex.

7.   A loop may be degenerate or it may consist of many loop edges.

8.   A loop is a boundary for a connected region of a face.

Topology

FIG 4-25
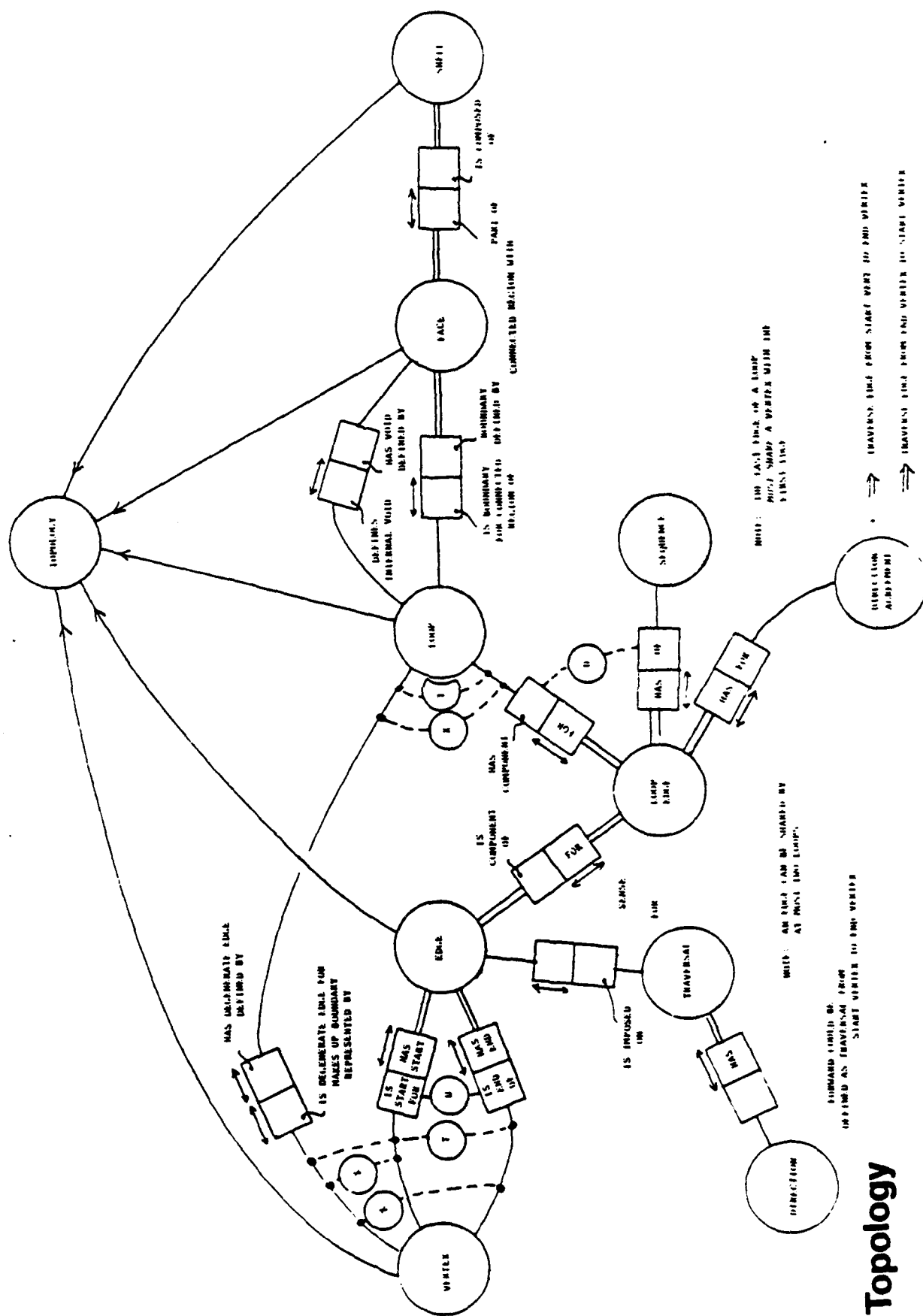
5.0 Lessons Learned

5.1 Use Of Several Languages

Most of the lessons learned during the initiation work stem from the fact that several different modeling languages were used. (Recall that both the application area models and the resource models were allowed to take virtually any form, and that IDEF-1 models and formal language models were encountered, in addition to the NIAM and the DSL models that were generated within the Logical Layer.)

Because so many languages were used, there has been some concern about not having some agreed-upon common set of conceptual notions that we could use as a principal resource in modeling. We realize that each language has mechanisms for triggering these conceptual notions, but the mechanisms are different for each language and lead to much confusion and wasted effort when translating between languages.

Our study of data base semantics, knowledge representation, and language translation leads us to believe that the formalism of first order logic with some extension is a sufficiently neutral mechanism for building and expressing a common set of conceptual notions. First order logic has a simple syntax and agreed-upon semantic. With the extension of multiple domains (formally called sorts), the notion of object type can be expressed. With the extension of lambda functions (and a particular interpretation of them that allows them to become predicates), first order logic can conveniently and concisely express arbitrary complexity.

Our recommendation is that any language that is to be used in PDES should be reconstructable in the extended first order logic language just described and that the basic conceptual notions be limited to those of first order logic. Said another way, any language used in PDES should be an interpretation of first order logic.

The principal conceptual notions of first order logic are: individuals are quantifiable (they either exist or they don't), true or false relationships can exist between any number of individuals (but preferably not between more than two or three to keep the models nearly atomic), and arbitrarily complex functions can be formed involving any number of individuals. These should be our basic conceptual building blocks. Each language will have its own particular way of converting these notions to symbols. We recommend following the guidelines established by ISO TC97 in evaluating and comparing conceptual schema languages. (See Reference 4.)

In the initiation work, we underestimated the cost of developing high-quality application models and the cost of translating between languages. We were never able to attain true translation. Instead, most "translations" were reinterpretations of the domain of discourse. On the average, five translation iterations were required to achieve acceptable qualified models.

We learned that information models evolve like any other complex artifact. We encountered numerous cognitive styles within the application areas, each reflecting the personal preference of the modeler. Some preferred text-based models while others felt more comfortable with graphic-based models. The most popular graphic form was IDEF-1 and the most popular text forms were variants of PASCAL data declaration statements. We had particular difficulty modeling the content of the Presentation document because it appeared to represent abstract program structures.

5.2 Global Conceptualization

The most significant tasks in the initiation effort were global conceptualization and Global model building. Recall that global conceptualization is the act of discovering how disciplines are similar and then replacing areas of similarity in discipline models with generic entities from the resource models. Global models then relate each discipline model to its corresponding generic entities in the Logical Layer Model.

Approximately 60% of the project time was spent preparing qualified resource models (geometry, presentation, topology) and qualified discipline models in preparation for global conceptualization and Global model building. Only about 20% of project time was spent in actual global conceptualization and Global model building. The other 20% of the time was spent in creating the PDES specification. It should be clear from these percentages that there is a very high up-front cost in getting to the global conceptualization and Global model building stage. It also suggests that we need to concentrate more modeling expertise at the application level if we expect the Logical Layer to really do its work. It would seem more reasonable that the Logical Layer would spend 20% of its time doing resource model development and model qualification, 60% on Global conceptualization and Global model building and 20% on specification. It can be seen from our actual percentages that we did not spend the majority of our time on what was considered to be the most important Logical Layer tasks. We do recognize that in the initiation effort that considerable time was required to get the basic resource models developed. Any follow on work that expects to completely rebuild the resource models should expect a high up-front cost comparable to the initiation work.

63

No significant global conceptualization was achieved until a provisional topology resource area model was built. This model was borrowed from the work of IGES ESP project and served us well. Once we built the topology model we began to see many different kinds of expressions of connectivity in the discipline models that could be factored out and expressed in terms of pieces of the topology model. The results of our initiation effort indicate that both geometry and topology entities are foundational to almost all application models.

We did not find the presentation area to be foundational to any application. Presentation appears to be a separate structure which can be constructed by traversing the actual artifact structure. This does not diminish the importance of exchanging presentation information, however. Rather than including presentation as an integral part of any model, we have developed high-resolution (in the spirit of PHIGS) associativities to geometry and text.

Much more needs to be learned about global conceptualization. We were able to conceptualize only about 10 to 20% of each discipline model (meaning that we were able to replace 10 to 20% of the discipline specific entities with generic entities). We should face the possibility that significant global conceptualization may not be achievable. We should also think in terms of replacing large discipline specific structures with large generic structures (as in the electrical example) instead of just replacing single objects. There are some parallels between the work of PDES and the large body of knowledge in the field of automatic natural language translation. The research in this field has been highly inspired and has covered many years. Should we expect to see a similar time frame for global conceptualization? This means that perhaps we need to scale down our expectations for the near term. For example, we may need to concentrate on just getting good conceptual models of the discipline areas and not attempt global conceptualization for a few years until we have more knowledge.

5.3 People

Turning now to people issues, the most important role of the project was the discipline liaison. In developing the methodology we envisioned a human being endowed with super human qualities. This role had to be filled by a person recognized as being an expert in his application area, as well as being a modeling expert. We did not find these kind of people, and the fulfillment of the liaison role become a cooperative effort between a discipline expert and a modeling expert.

The liaison role is a critical link between the world of application knowledge and the process of global modeling. We learned quickly that acquiring application knowledge and modeling

64

it is very time consuming. We believe there are similarities between the capture of PDES application knowledge and the capture of general knowledge in the field of knowledge representation, where the automatic translation of natural language into conceptual models is an intense research area. Perhaps we need to become more aware of work being done in general knowledge representation.

Modeling is new to most people. _We have experienced some delay in our project so that people could become more comfortable with modeling concepts._ People are still making the shift from thinking in terms of tables (which, incidentally, is not their natural mode of conceptualization) to thinking in terms of objects and relationships. Most people desire to abstract away the detail in structures, and the modeling aspect of the initiation effort has forced the application area modelers to reconsider the subtle structures embedded in their application knowledge. We need to be much more aware of the work done in the area of cognitive psychology when trying to determine a style of representing application knowledge. PDES represents a new approach that starts with humans and their knowledge, not with computers.

5.4 Cognitive And Specification Models

One of the most significant things we discovered in our effort was the distinction between the cognitive model and the specification model. (Recall that, roughly speaking, the cognitive model is the one that provides the environment in which to think, and the specification model is the one in which the product of the thinking is packaged.) _We learned the following very simple lesson: first the model is conceived and then it is packaged - don't attempt to conceive and package the model at the same time._ The specification model itself is not to be thought of as an implementation model, and should not be any less powerful in expressive power than the cognitive model.

In terms of the particular techniques used in the initiation work, the NIAM model is the cognitive model, and the DSL model is the specification model. We believe that this decoupling between thinking and specification is essential.

5.5 Modeling

We have learned some things about the state of our modeling art within the Logical Layer. Some of these things are expressed here in terms of the NIAM modeling technique that was used to construct the global models.

One of our observations is that many of our NIAM conceptual models contain relationships that are poorly named. We probably have too many "has-of" relationships. This is an indication that we are modeling the structure of objects, not the meaning of

65

their relationships. This may have something to do with the basic roots of our experience, which for the most part has been program development and database design. On the other hand, the field of knowledge representation seems to be almost obsessed with identifying, naming, and classifying relationships. Should we follow suit?

Our data base background and the types of modeling languages we are accustomed to (particularly IDEF-1 and NIAM) have caused us to focus our thinking about relationships into two general areas: functional dependency and cardinality. These two basic notions of relationship may not be sufficiently powerful to model PDES artifacts. On the other hand, the only notions of relationship that seem to have universal acceptance are those which can be expressed as mathematical relations. The notions of causation and definition, for example, do not have agreed-upon meaning.

There was some concern early on in our effort that the weakly typed NIAM language would not be able to model the complexity of the artifacts we expected to be modeling. This was not the case. Strong typing makes describing complexity more convenient and more concisely presentable to a model reader. Weak typing is extremely powerful when then structure of an artifact is being explored and there is not yet a desire to bind certain objects and relationships to a type structure. Weak typing promotes amorphous model structures that feed the cognitive process. Strong typing leads to more concise definition and named abstractions which are good for model packaging.

5.6 Translation Between The NIAM Cognitive Model And The DSL Specification Model

Recall that the translation between the NIAM global model and the DSL specification model is currently a manual process. We have experienced certain difficulties in the initiation work in carrying out this process.

In the DSL language it is obvious that the language developer saw a significant distinction between a mandatory relationship and an essentially defining relationship. It is the notion of essentially defining relationship that gives cohesion to most DSL entities. (An example of a mandatory relationship that is not essentially defining is a relationship enforced, say, by company policy which states that an employed must be referred to formally by an employee number. Certainly, the relationship between a person and his employee number is not essentially defining.)

Now, NIAM uses only relationship notions that can be built from constructs that have the classical semantics of sets and mathematical relations between sets. So, in NIAM any other subtle notions of relationship are hidden in the interpretation of the name of the relationship. Therefore, in translating between NIAM and DSL, one must carefully understand the meaning of all the relationships.

We agreed that we would provisionally augment the NIAM graphical notation by adding a "dot" on all relationships that we felt were essentially defining relationships. Our experience so far has been that this greatly reduces the time necessary for the specifier to understand and place the NIAM model into DSL.

We were, however, extremely reluctant to make any changes to NIAM syntax as its power is in the leaness of its syntax. We have reservations about ever formally agreeing on a definition of the notion "defining relationship." There is no mathematical formalism that establishes the meaning of "defining relationship."

A crucial issue in building the specification model from the global cognitive model is the development of a rationale for grouping the global cognitive model. We recognize grouping as an act of preparing the global cognitive model for specification we do not recognize grouping as a cognitive act.

5.7 Summary

Our experience has led us into a broader knowledge of conceptual modeling. Perhaps the most important lesson learned is that we had to remove the blinders of our own experience and be able to assimilate the knowledge of other people on the project and people in related research fields.

6.0 Critical Issues and Recommendations

This section will deal with general issues and specific issues. General issues relate to foundational issues of the PDES Logical Layer such as how people and the PDES working environment relate to languages, conceptual formalisms, and the real world to be modeled. Specific issues derive from specific problems encountered during the application of the methodology and their impact on the direction of the Version 1.0 Logical Layer Activity. This section concludes with recommendations.

6.1 General Issues

Regardless of the specific direction in which the PDES Version 1.0 effort heads there will be fundamental issues that must be dealt with. This section hopefully will prevent us from concentrating too hard on just the Initiation Experience.

6.1.1 People

We need to categorize the roles of people into three broad areas: 1) those who are developers of methods, tools, and languages; 2) those who support the application of methods, tools, and languages; and, 3) application experts. This is a rough breakdown for the purposes of exposition.

With regard to the developers of the methods, tools, and languages for PDES Version 1.0, we should seek to find diversity in experience and attempt not to focus too much on any one particular style of method, model, or language. PDES itself is quite diverse and it may require within the single method of PDES a diversity of approaches. An example of diversity is to have developers from the fields of information engineering, knowledge representation, language development, natural language translation, and cognitive psychology.

Methodology support persons must be willing to adapt to a new way of doing things. It is hard to think that the PDES method of the future will be just a simple adaptation of existing methods. These people will have the key responsibility to acclimate others to the method and in this sense the support people are quite critical. Obviously these people should already have some existing background in the fields of data base, knowledge representation, or modern programming languages. We cannot expect application experts by themselves to keep a new method alive. That should not be their key role.

Application experts drive the method. Nothing in the method can replace their expert application knowledge. There is some

68

question about how involved in the method these people should be. We don't want to dilute their energies or throw artificial barriers in their way (they may possibly perceive that any modeling language is an artificial barrier). Actually we should not speak in terms of a single expert but rather a group of experts who cooperate. This gives more options. If the group is large enough it may be possible to find a few individuals to become more involved in the method and serve as representatives to the others. The principal issue of the knowledge representation field will always be with us: how do we get the knowledge from the expert? There seem to be three main approaches: 1) have a knowledge representation expert work with the application expert; 2) teach knowledge representation techniques to the application expert; 3) develop powerful natural language translation software that can automatically build conceptual models during a machine-expert dialogue.

6.1.2 Environment

The environment stages the development of the PDES standard. It would include the basic methodology which frames the whole process of development, giving it order and assigning skill and management roles. It would also include generalized software that will allow the integration of specific graphic tools, modeling languages, and translation capabilities.

Much remains to be done in terms of developing the environment. This is quite a critical issue, as we know that these kinds of environments (particularly the distributed ones) are just now coming into their own. Although we have seen CAD/CAM equipment used in "spot productivity" modes, we are still waiting for the synergistic environment. The "information engineering environments", long overdue, are rapidly becoming complex products running on small desk top computers or workstations. At least these products should give us a vision of a modern PDES environment. However, we must characterize the PDES development environment and resist the urge to adapt an existing one. Admittedly, this is an idealistic approach but it should not be counted out. (The section on specific issues will explore other alternatives not quite so idealistic.)

6.1.3 Models And Languages

Diversity is the single most critical issue in this area. Diversity stems from the tremendous breadth of PDES application and the wide variety of experiences of the people associated with those applications. No attempt at building a PDES development environment will escape the impact of this issue. It is no harder to conceive the need for a variety of representation techniques than it is to conceive the need for a variety of people to run a company or a toolbox full of different tools to build a house.

69

We fully recognize that the need to have model and language diversity seems to run against the need to build global models in a single language. What we see here is almost the equivalent of the problem that IGES/PDES is attempting to solve in the evolution of a product. Each life cycle stage seems to have its own representational needs but each desires a common way of communicating that representation in an application free mode.

The need for diversity of language and representation leads naturally to the development of a common representational foundation that is "neutral" to all application representations and languages. PDES must not get into a war of languages over the need for diversity. Without a common foundation, pairwise comparisons of languages will dilute our efforts.

<u>We have done considerable research during the initiation effort to discover the "right" foundation formalism. We believe the language and notions of formal logic are sufficiently powerful to serve as this foundation.</u> This is not just our opinion. It is also shared by others in fields where diversity and desire for unification are also occuring such as data base, knowledge representation, and linguistics. We would hope that such pragmatic issues as weak typing versus strong typing, text versus graphics, atomic versus non-atomic, procedural versus declarative, and representation styles such as rules, nets, frames, and objects do not erode our desire to create this essential foundation. This foundation is the "IGES within IGES".

6.1.4 Broad Knowledge

This issue is really just a statement of a common theme running through the discussion of the other issues, but is significant enough to demand special attention. We must be careful in developing the PDES that our knowledge becomes too narrow and focused. For instance, we could use traditional data base development methodology as a start point for developing and executing an initial PDES method, or we could use our knowledge of strongly typed programming languages, or try the latest tricks in logic programming.

The PDES initiation effort was probably too focused on data base semantics (more about this later), but because of the time frame it was justifiable to take an approach and run. The Version 1.0 effort should assimilate as much knowledge as it can in the time frame allowable. It is worth consideration that building a broad knowledge base and perspective should actually drive the schedule.

6.2 <u>Specific Issues</u>

6.2.1 People

Although there was a large family of participants in the Initiation effort, the group that ultimately developed the method and initial content of the logical layer was rather small. Members of the group were separated by hundreds of miles and had very limited personal contact with each other. This is probably normal for standards efforts. But PDES is a different kind of standards effort that may demand a shift from the norm. The amount of time for which companies will volunteer their people will have a direct impact on the quality of PDES results.

From our experience, the size of the application groups was adequate in most cases, but there were some groups of just a few people. A size of fifteen or so seems appropriate. It is questionable how big the logical layer group can be and still function without interferring each other. Obviously there are some ground for specialization. The Phase 1 methodology efforts in qualification can be independent efforts with, say, one modeler dedicated to each application area. Phase 2 must be a cohesive group of probably not more than 7 people dedicated to pushing the leading edge of conceptualization. We recognize the need for "spot" consulting support from academic and industrial experts in the field. We emphasize "academic" support which was almost totally missing from the initiation effort.

We have already written about our experience in allowing unlimited selection and usage of languages on the part of the application area. Perhaps we need to limit this language set to a qualified five or six. Perhaps if more individuals were assigned to qualify application models, each qualifier could be teamed up with an application area using a modeling language that the qualifier is already familiar with. In other words, try to limit the qualifier to just two languages: the standard model language of the logical layer and a single application language.

Participants in the PDES effort must believe that the methodology will work and play their assigned role(s). Job specialization may be the "revolution" within the information revolution. We obviously need to learn from very-large proje s and how work is divided. The initiation effort gives a good start in classifying the kinds of roles to be played.

People must agree on certain styles of modeling the qualified schema as this will ease the burden of the Phase 2 conceptualizers. This brings up a serious issue: finding enough people who model well in any modeling language who are willing t adopt a common style. Because we had basically only one model qualifier in the initiation effort, we were not able to gain experience in this area.

A significant issue regarding people is the complexity of language with which application persons will be willing to work. If the language is weak then we lose valuable application knowledge because the language cannot express it. If the language is too rich we lose valuable application knowledge because experts won't learn it. We have applied the "walk before run" principle in the initiation effort and have settled on fairly simple classes of languages that are easy to learn and use. We want to "walk" now, but we perceive that we may have to "run" later. We are not convinced that languages richer than the ones we are already using will pay off in the near term.

6.2.2 Environment

The environment for the initial effort was minimal. It was basically composed of a description of a methodology, an allocation of languages to phases of this methodology, and a few informal agreements with the application layer. Early on, there existed no documentation standards, such as the kits in the IDEF methodology. Later in the effort, a standard kit for application model documentation was suggested.

Although the methodology was simple, it did serve to give us guidance. The basic three phase methodology should be adequate for the PDES Version 1 Logical Layer work; however, other aspects of the environment must be further developed. A model configuration management subsystem should be developed. Computer assisted graphics for building models is badly needed to replace the pen-and-ink approach of the initiation effort. A computerized conceptual dictionary with a powerful query capability will be needed in the PDES version 1 effort.

We envision an environment consisting of a network of workstations (or at least personal computers) with a central model administrator. Most likely the global PDES model will become large enough that it must be stored on a mainframe. We are just now seeing examples of information engineering workstation environments appearing in the marketplace. This is encouraging, and we suggest that these new products be investigated. Graphics support for information modeling is also becoming available. A critical aspect of the PDES Version 1.0 work should be to investigate these emerging capabilities.

A critical issue that affects the development of the environment is a thorough characterization of PDES information. Is it mostly numeric or is there a significant symbolic component? What is the mix of administrative, engineering, and shape-related information? Bascially there is no "strategic" picture of PDES information.

The methodology used in the initiation effort did not have a strong project management component. It only suggested the technical content with plausible deliverables between phases. The methodology does not suggest a way in which it may be operationally placed within the IGES/PDES organizational structure. The initiation effort did not attempt to determine what kind of impact a methodology would have on existing IGES/PDES operating principles and protocol.

6.2.3 Models And Languages

A critical issue is whether to allow an unlimited number of application modeling techniques. The PDES Version 1 effort needs to consider at least three alternatives: 1) unlimited modeling techniques 2) a single standard technique or 3) a limited set of techniques. The cost of living with several languages is quite high. Because most people are not yet culturally locked into any given modeling technique (although entity-relationship techniques seem most popular), we should take advantage of the flexibility that we still have to find the best language(s).

We strongly recommend using graphics-supported modeling techniques at the application layer and in phase 1 and phase 2 of the logical layer methodology (the cognitive portion). We are aware of the limitation of graphical techniques in voicing constraints and suggest that mixed graphic/text techniques be chosen. No modeling technique should be chosen that does not have a single uniform text language in which any construct of the modeling technique (whether graphic or text) may be expressed in a computer sensible form.

It is strongly suggested that a single (i.e., the same) modeling technique be used for phase 1 and phase 2 of of the methodology. This technique should be a detail-exposing one. It should allow completely ungrouped modeling to occur. Particularly, it should not force the phase 1 and 2 modelers to use the relational model, or the entity attribute relationship model, as popular as these models are. It should exploit natural language expression of relationships and constraints as much as possible.

The Version 1.0 effort should be concerned about the expressive power of the application modeling language(s). A language that is expressively weak may only push the application modeling work down into the qualification phase of the logical layer. A language that is too strong may be perceived as an artificial barrier to the application expert. A compromise is to choose a reasonably strong language with graphical support and let the expert express in English what he cannot express formally. The mappings between a constraint in English and the same constraint in a formal constraint language are almost never straight forward even for the simplest of constraints. We

73

definitely see the phase 1 and phase 2 workers as being experts in FORMAL constraint articulation. If the worker is not capable of converting natural language constraints into well-formed sentences in predicate logic, he is probably not expert enough. We even suggest the possibility that first order logic be a staging area for expressing constraints even before they are voiced in a specific modeling language.

This leads us to another issue: the need for a common conceptual formalism. PDES cannot become a circus of languages with each language having its own set of proponents. The formalism serves as common ground for the understanding of two modeling languages and also can be a standard neutral form in the translation between two dissimilar languages. The ISO TC97 guidelines for conceptual schema languages is an example of the use of logic as a reference for evaluating languages. We strongly recommend that formal logic be used as the common conceptual formalism. As suggested earlier, we suggest specifically the use of the first-order predicate logic language as the standard neutral language to be used in all cases of arbitration of meaning and comparison of modeling languages. We further suggest that before any language is used that it be shown how that language can be reconstructed in first-order predicate logic. We realize the limitations of first-order logic and see the critical need to extend it to be able to express arbitrarily complex abstractions ranging from simple types to complex generalization and aggregation abstractions. We have observed a similarity between the complex utterances of natural language and the complex artifacts of PDES and suggest that we investigate some of the same techniques being used to convert natural language to conceptual models. Lambda abstracting is such a technique.

The initiation effort basically borrowed data base modeling techniques to do most of its work. This approach has only partially captured application semantics and should not be perceived as the ultimate modeling solution. The PDES Version 1.0 effort should be on a path to capture full application semantics. This suggests a broadening of our modeling approaches to include those being used by the field of knowledge representation. Choosing languages which are expressively weak will result in incomplete formal models of applications areas (natural language patches will most likely get lost in the process). It is critical that PDES Version 1.0 look beyond data base languages.

Neither IDEF-1 nor the MIML subset of NIAM is considered powerful enough for the Version 1.0 effort. Neither has the expressive power of the first-order logic. IDEF-1 is extremely weak in declaring constraints. The MIML subset of NIAM is somewhat stronger. A language exists to extend the constraint power of the MIML subset of NIAM, but our understanding is that

74

this language is proprietary. If neither IDEF-1 nor the MIML subset of NIAM can be extended, we suggest looking for a new language. The developers of DSL are presently working on extensions to extend its expressive power. We suggest, however, that DSL be used exclusively as a specification language in the Version 1.0 effort. It must have the expressive power of the Phase1/2 language.

6.2.4 Broad Knowledge

The issue here is that those involved in the Version 1.0 effort must have a broader knowledge of modeling techniques and languages than did those in the initiation effort. Because of schedule constraints, the initiation effort concentrated on using the modeling knowledge its members already had in order to quickly test a methodology and build content. The Version 1.0 effort should not follow the same course. If maintaining schedule within PDES is the driving motivation, we can expect to see a narrow focus on modeling technology and a resultant loss of capture of application meaning. The capture of application meaning must be at the heart of any PDES effort that seeks to follow the development guidelines of the second PDES Report.

A specific example of adopting a broad-knowledge attitude is the acceptance of logic as formal foundation for modeling. We have already seen this broadening aspect in the field of data base, knowledge representation, and natural language translation. The trend seems to be that workers in these fields are understanding the importance of formalisms. We believe that PDES should follow suit.

A specific area where we are lacking formal thinking is in the development of the global models. We are presently using informal terminology such as "resource model" and "user model." In the language of logic and axiomatic systems, concepts that are defined as true (the elementary building blocks) are called axioms. Deduction is then used as a powerful tool to build new concepts. A broadened knowledge of conceptual modeling should accept the principles of logical deduction as a necessary tool. Deduction adds a generative flavor that is missing from our present repertoire of basic concepts.

7.0 Summary Of Lessons Learned And Recommendations

This summary section proceeds from two fundamental points:

1) it is the information in the application models that is to be exchanged.

2) the needs of PDES dictate that these models are to be computer-sensible, and able to be exchanged with their structure intact.

Lesson Learned:
There is a shortage of good modelers.

Recommendation:
Encourage, sponsor, publicize, etc. programs that promote modeling.

Lesson Learned:
Lack of modeling expertise can delay schedules.

Recommendation:
Insist on quality models over fast schedules.

Lesson Learned:
The Logical Layer work of conceptualization and integration proceeds first by perception, concept discovery, and exposition of detail, and then by specification. The decoupling is essential.

Recommendation:
These two types of activity should be supported by cognitive and specification models, respectively. The cognitive modeling technique should have a graphics component. The two models should have equivalent descriptive power, and both should have a computer sensible representation. The two models should share a common foundation based in first order predicate logic.

Neither IDEF-1 nor the MIML subset of NIAM is considered powerful enough for the Version 1.0 effort. If neither can be extended to have the expressive power of first order logic, a new language should be sought.

Lesson Learned:
Global conceptualization and integration is a human intensive activity that cannot be accomplished in large groups.

Recommendation:
Local conceptualization must occur systematically at the Application Layer before expecting the Logical Layer to do global conceptualization and integration

Lesson Learned:
The cost of living with several languages at the Application Layer is quite high.

Recommendation:
Either mandate a single technique or limit the set of techniques. If more than one technique is allowed, the techniques allowed must share a common conceptual formalism. The appropriate formalism is first order predicate logic.

Lesson Learned:
The discipline liasion role is the most important role of the project.

Recommendation:
Find the application experts who have an interest in modeling, and cultivate that interest.

Lesson Learned:
Don't automatically assume that traditional data base techniques will suffice when dealing with our data exchange problem. We couldn't model everything we wanted to in a useful computer form because the modeling techniques we had were geared to administrative type structures, not more complicated PDES type artifact structures.

Recommendation:
Don't attempt Version 1.0 in this state of affairs, because the needs of PDES call for computable models.

Lesson Learned:
    There is a difference between following a development
process for writing a data exchange specification, and actually
using that specification to exchange data in the manner intended.
Data exchange involves user data bases and vendor translators.
Many data exchange issues have not yet even been raised, much
less resolved. The PDES effort in some ways is a multi-company
research effort based on volunteer labor.

Recommendation:
    Begin immediately to learn what data exchange in a PDES
environment might involve. For example, what the structure of
user data bases must be, what translator capabilities will be
required. Achieve an understanding that the factor of the
unknown may affect schedules in ways unsuspected as yet.

Lesson Learned:
    We were significantly affected by the lack of computerized
tools particularly graphics support for creating modeling
diagrams. We had no full-feature conceptual dictionary.

Recommendation:
    Develope a specification for an information modeling
development envrionment and procure the environment before
serious effort at PDES V 1.0 is started.

Lesson Learned:
    We believe that the methodology developed in the initiation
experience is a proper framework for development of PDES content.
Although some phases and techniques must mature, we believe there
are no major missing pieces in the methodology.

Recommendation:
    Use the methodology as developed in the initiation effort as
the framework for developing PDES V 1.0 content.

REFERENCES

1. The Second Draft Report Of The Ad Hoc Committee On The Content And Methodology Of The IGES Version 3 (The Second PDES Report), by Kalman Brauner and David Briggs, The Boeing Commercial Airplane Company, November 12, 1984.

2. IFIP WG 8.1 - "NIAM: An Information Analysis Method", by G. M. A. Verheijen and J. Van Bekkum, _Information Systems Design Methodologies: A Comparative Review_, North Holland, 1982.

3. Concepts And Terminology For The Conceptual Schema And The Information Base, J. J. van Griethuysen, Ed., ISO/TC97/SC21/WG5-3 Report SC21-N197, March, 1982.

4. Assessment Guidelines For Conceptual Schema Language Proposals, J. J. van Griethuysen, M. H. King, Eds., ISO/TC97/SC21/WG5-3, August, 1985.

Appendix A

<u>Comments By The Application and Resource Modelers</u>

1

Comments By The Application And Resource Area Modelers On The Modeling Experience

This appendix consists of statements by task leaders or other persons involved in the developement of the various models used in the initiation effort. The statements summarize their reaction to the use of modeling techniques in carrying out the work for the initiation effort. The contributors to this section are also the contributors to Section 3.

Wireframe Geometry Modeling Experiences With IA - Edward Clapp

Let me start by saying that this is an interim and purely personal assessment of the utility of NIAM modeling techniques in the building of PDES/STEP. I have not yet discussed my reservations with either John Zimmerman or Paul Thompson. I have very high regard for both of them and their efforts and feel that it would be useful to work out my thoughts with at least one of them.

This is a portion of an effort on my part to try to critique the PDES efforts to date. A number of people have done a lot of inadequately appreciated work which deserves both careful review and analysis. This paper is neither, being merely an attempt to indicate some concerns I have about the use cf NIAM in PDES.

There are some positive things I can say about NIAM. Two of the purposes of a formalism are to help organize and clarify. NIAM does this in working with PDES constructs. The visual approach may well make it easier to get an idea for what is going on, much as state diagrams are easier to understand than equivalent descriptions of finite state machines. This is helpful both for the people working on a particular application and for the people wishing to ensure that the constructs of one application are not merely pieces of some other application indisguise.

Now for the "howevers".

Two of the major criticism of IGES made by the user community were the size of IGES files and the length of processing time. While it might be reasonable to decide that PDES be inherently less efficient in these ways, that decision and the reasons for it should be made explicitly. NIAM was designed to decompose data. This has led to ↳ proliferation of entities which would be expected to lead to increased file sizes and processing times.

Just as state diagrams are not used in proving theories about finite state machines, it seems to me that the correct place to be formal about what is being said is in the DSL and

1

that the people working at the application level are the ones who need to make the formal statements about the data.

The separation of file syntax from the application by two (NIAM and DSL) levels makes it much harder to introduce processing efficiencies.

Lastly, the extra process between the application and the physical layers makes it much more difficult to incorporate changes. In the initial stages of building PDES it would be useful to be able to quickly build and test prototypes. Applications writing to the DSL would be able to do so far more efficiently than those writing to NIAM.

Electrical Schematic Modeling Experiences - Curt Parks

Prior to the PDES initiation effort, model development within the Electrical Applications Subcommittee had progressed to the point where the benefit of information modeling for application data requirements and for data relationship proofing, given a target application environment, could be verified. The environment was the IGES physical format, and the data content was that of several known CAD systems used for the design of Printed Wiring Assemblies. (See Section 3.3.3) The model also served as a guide when test cases were developed following the definition of IGES extensions. The results of this experience with modeling are extremely significant. The Electrical Applications Committee will no longer propose data entities for inclusion in a specification without first constructing and validating a logical model.

The schematic model for PDES had several additional aspects:

1. The model was to be independent of any application environment or physical data format.

2. The more expressive modeling provided by IDEF-1 Extended would be used.
3. Computer modeling aids were used or evaluated.

4. The resulting model would be subject to integration with other application models.

The initial model required only a few hours work by two people who were familiar with the application and the new IDEF-1 Extended rules. Either a related application model (as in this case) or a collection of entities extracted from reference reports serve well as input. Working with "collections" of dependent and independent entities ranging from 5 to about 15 on each page was quite effective. The model walk-throughs were made much easier by the groupings, and no need was found to further reduce the entities-per-page after consensus was reached.

2

The development from the initial model consumed the bulk of time, and was mostly due to human factors, given 5 to 6 meeting sessions per year and a changing membership. Each meeting session required a review of the modeling language, and the criteria for entities in the model. A frame of reference was required wherein the members had to conceptualize the information found in the application as opposed to the more vocationally revelant view of its usage. Following an overview of the previous model, and the changes made to it, problem areas could be identified and discussed. After each meeting all agreed-upon changes or additions were incorporated, and, following the IDEF "kit" procedures, the new package was sent to attendees.

The model kits were also made available for comment to people and organizations not attending the work sessions. No comments were received from such sources, except the model integration team. The EAC is presently exploring ways of bringing a wider range of experts to the model reviews, or holding the reviews at gatherings of users and experts. Eventually, for the assurance of the acceptance of PDES by vendors and users, they must (to whatever degree possible) be brought into reviews for both application and logical layer consensus.

At present, the modeling methodology is seen by the EAC as the best means of developing a logical schema, but more people must understand what logical models are and how they are developed. The use of the picture-like language has greatly increased meaning to people during reviews. Further, those pictures must be drawn by people to insure human "readability". The computer-automation was effective only as a normalization and verification tool after the model was created. Interactive graphics computers work well for producing "polished" model views, but models produced by automation from model data were not suitable for reviews or kits. The layout produced by the automated software seems too spread out for convenient viewing.

During the intergration phase, the use of a different modeling methodology had both good and bad aspects. The conversion forced a complete "what was meant" review of the application model views and glossary. Areas were thereby discovered where entity/attribute names could be adjusted to align with other models. The conversion was, however, an extra step which tended to remove the results away from further review by the application committee. The original model language seems to convey something akin to poetic essence which is often lost when translated, even though the content is retained. When the model is further reduced (e.g., to Metamodeler output or a DSL) to text, a group review becomes all but impossible.

Reduction of a model to English sentences seemed to help only when done verbally as part of a model walk-through. Again, pictures can retain interest where sentential facts are dry to the point of disinterest.

The process of integration would have been greatly aided if a logical layer product model had been developed first. Each model needed a focus in the form of a PART, COMPONENT PART, PART VERSION entity structure. This is now being developed for PDES and will greatly help future application modeling. In turn, the product model should be constrained by a definition of domain. The domain question is vital to the entire PDES effort. Is a product in PDES format going to be able to produce, or to replace, a drawing or blueprint? Is it to include the entire product life-cycle (another dimension of product definition)? Or must it convey the as-designed product against which an as-manufactured and as-maintained product can be compared? The follow-on electrical product model is being developed in a narrow domain (no blueprint, as-designed) to avoid the multi-dimensional aspect possible and the related confusion. The resultant model can be expanded later given a consensus on the larger domain PDES product model.

Tolerancing Modeling Experiences - William B. Burkett

1. Introduction

This article is a summary of the experiences garnered by the Tolerancing Application Group (TAG) from the PDES Initiation Task. It is intended to represent the viewpoint of the TAG as a whole, but some of the thoughts expressed herein are those of the author and may or may not reflect the feelings of the entire committee.

One of the objectives of the PDES Initiation Task was to serve as an educational prototype of a working PDES organization. The intent of this article is to report on the lessons learned" from that task. The topics covered here will start with the generl procedure followed by the TAG and it's subsequent interaction with the Logical Layer Committe (LLC). This will be followed with a discussion of the major problems encountered, a list of iportant observations made, and then a summary of the entire process.

A description of content of the work of the TAG can be found in Section 3.4.

4

2. Procedure Followed

The TAG was set up as an independent Application Layer Group to develop a reference model of the information required to tolerance a 3-D geometric model according to the ANSI and ISO standards (ANSI Y14.5M 1982, ISO 1101, 1660). The model produced was then delivered to the LLC and a review cycle involving the LLC and the TAG was started to monitor the cnversion/integration process. Once the integrated model had been validated by the TAG, the TAG's task was considered complete.

The TAG first met in early 1985 and spent eight months developing the first draft of the Tolerance Reference Model (TRM). Members of the TAG were Ron Bale (Caterpillar), Bil Burkett (McAir), Bob Colsher (IGES Data Analysis), and Spencer DePauw (Caterpillar). After the TRM was delivered to the LLC, Mr. Colsher and Mr. Burkett spent the following four months supporting the LLC's conversion and initial integration process. During the last three months of the TAG's activity, Mr. Burkett supported on the LLC's integration process and reviewed and commented on the tolerancing subset of the integrated model.

There were three tasks undertaken by the TAG before an actual modeling was started. First, a charter was prepared outlining the scope of the TAG's activities and offering a justification for the approach taken by the TAG. The second task was a review of the work done by the PDDI project on the subject of tolerances to determine the suitability of that work to the needs of the TAG. The last task was the selection of a modeling technique for the reference model. The TAG used a modified version of the IDEF-1 technique which was actually an extension to enhance the ability of IDEF-1 to represent information (these extensions were considered as "author conventions" in IDEF-1 terminology and originated from the PDDI project).

The development of the model consisted mosty of reaching a consensus on the interpretation of the standards and how that interpretation could best be represented with the modeling technique.

Once the TAG agreed on a "finalized" version of the TRM it was delivered to John Zimmerman (Allied Bendix) of the LLC. It was the LLC's job to convert the reference model to a binary Information Analysis (IA) model and integrate it with other application reference models. From the integrated model, entities were then defined in Data Specification Language (DSL).

As part of the review cycle, there were several in-person meetings and numerous telephone conversations to verify/validate that the meaning expressed in the TRM was maintained as the model was converted to the IA model and integrated with other models. Each time a new version of the converted/integrated model was

produced, a copy was provided to the TAG for review. The model was finalized (as much as possible) in early 1986. The TAG did not review the DSL specification formulation of the tolerancing information.

The conversion and integration process discovered some mistakes and unclear thinking in the orginal TRM. The TAG made use of these discoveries to revise and correct the model. In all, three versions of the TRM were created.

3. Special Considerations

There are several factors which influenced the modeling processt that should be noted. First and most importantly, all members of the TAG were very well versed in the subject area (the ANSI and ISO standards). Second, Mr. Burkett was a member of the PDDI project and was responsible for the approach to tolerancing taken by that project; Mr. Colsher is chairman of the IGES Drafting committee and responsible for the IGES approach to dimensioning and tolerancing. Third, most members of the TAG were experienced with data modeling techniques before work was started.

4. Problems Experienced

The biggest problem, obviously, was the amount of time required to satisfactorily complete the modeling/integration process. The experience of the TAG and LLC held the time spent to a minimum, but the entire process still took a considerable length of time. The TAG spent about as much time in the review process (with the LLC) as it spent to develop the origina TRM. The amount of time that the LLC spent on the conversion and integration process outside of the review process is difficult for the TAG to gauge; a guess would be that the review process constituted about ne-quarter of the time of the LLC spent on the model.

Another problem that the TAG experienced during the conversion/integration process also came up to a lesser degree during the development process. By modeling the same information using two different modeling techniques, it was very clear how difficult it is for any diagramatic technique to represent understanding of a particular subject domain. Human language (and human understanding) is imprecise and inexact - diagrams are not. It was very difficult to make the mental leap from the understanding that the TAG had of the ANSI and ISO standards to an expression of that understanding by means of a diagram.

The last major prolem experienced by the LLC (it didn't really affect the TAG) was the fact that the TRM was very large, looked complex and was, therefore, intimidating. This was

6

probably the result of overcompensation on the part of the TAG. Because of past experience with modeling techniques of limited expressive power,the TAG attempted to get everything that they knew about the subject domain into the model without resorting to notes (such as the fact that a Flatness tolerance applies only to planar faces). As a result, the model was semantically richer (expressed more meaning) than a typical reference model, but it was much bulkier, too. Once the LLC became familiar with the model, the size did not turn out to be a problem because the model was composed mostly of simple, similar, consistent concepts.

5. <u>Important Observations</u>

The following is a list of important observations made during the entire modeling process. It is basically an unstructured set of observations reflecting problems, benefits, solutions, ideas which seemed to work best, problems with the content of the model, and anything else which seemed to be relevant.

o A drafting representation of the tolerancing information based on conventional graphical symbols (witness lines, leaders, text position, etc.) cannot be generated solely from the information contained in the reference model. Drafting is a presentation concern which is "further away" from the geometric model than tolerances.

o The Logical Layer played the role of a model validator, rather than a conceptual modeler. Most of the conceptualization was done at the Application Layer.

o Truly functional aspects of the PDES integrated conceptual model did not at first drive the Logical Layer development work. The PDES geometric entities at that time did not have the necessary constructs to support the tolerance model (such as topological constructs). (Instead, much time had been spent on geometry and presentation reference models without regard to the needs of the applications.) However, the needs expressed in the TRM were eventually satisifed by the work of the LLC.

o The TRM was very "proactive". It described the subject domain and contained well defined "links" to the integrated model. It did not define geometry, yet it made certain demands of it (i.e. topology).

o The fact that two subject experts were present during the review process proved to be very valuable. Multiple points of view allowed ideas to be played off of one another and resulted in a more "accurate" model which was arrived at more

7

quickly. A single expert presents only a sigle viewpoint, so
the double coverage was very beneficial.

o The TRM was well thought out beforehand so there wasn't much
  ad hoc application modeling being done during the review
  sessions. This allowed more time to be devoted to the
  understanding and conversion of the model. This is not to
  say that the review process (conversion and integration)
  didn't uncover some bugs in the TRM; that process did prompt
  several revisions of the TRM.

o Knowledge of modeling was a key ingredient to the
  communication between the TAG and the LLC during the review
  cycle. By the time the last reviews were taking place, the
  TAG was reviewing the LLC's IA models.

6. <u>Summary</u>

     Overall, the process followed by the TAG and the LLC seems
to have worked quite well and with a little experience (and more
than a little resources) could be used on a larger scale. Four
words succinctly summarize the process: Compose; Decompose;
Integrate; and Recompose.

     The use of two data modeling techniques to support this
process was more a boon than a bane. Each technique has inherent
problems with expressiveness and understandability, but the use
of both minimize the effects of the problems. The conversion
process actually makes the models better because the weaknesses
of one technique are compensated by the strengths of the other,
thereby uncovering bugs in the original model. Perhaps in the
future when everyone is an expert modeler, one technique will
suffice. Curently, we don't understand either technique well
enough to fulfill our needs.

     The conversion process forced the subject experts to dot
their "i"'s and cross their "t"'s. It was realized that the
understanding of the subject domain is inexact, incomplete and
often contradictory. The precision of the modeling technique
forces a formalization and rigor in the understanding of
information that words cannot convey.

     The biggest weakness of the entire process and the source of
most of the problems are the tools used to support it. The tools
we have are good and are a big improvement over what was
previously done, but they still have problems which make them
difficult to use. Automated tools might help, if only for
generating graphics.

     The only immediate solution for the problems encountered
during the work, of course, is experience. This problem is a
problem of any modeling technique and the best model is at best a

8

best guess. The biggest obstacle that a new modeler must overcome is the flustered feeling that the ambiguity of the job can cause. The solution is simply to ignore it; there is no "right" way.

As for the time resource problem, it will become less of a problem as people become better modelers and better understand their subject domain. The entire process will still take a significant amount of time, but it could be considered an investment in a common understanding of an subject domain. A common understanding improves both the completeness and the ease of communication, and that is what we're all after, right?

FEM Modeling Experiences - W. R. Freeman

I applaud the overall approach to the creation of PDES, with the three layer architecture, and the use of application experts to initially define their "universes". Combing this with the integration at the logical layer, and definition of the physical file format AFTER the logical layer is completed gives a feeling of a solid, sensible methodology. This methodology is innovative, and if carried out as presented should result in a consistent standards document with a relatively small, non-repetitive list of flexible entities. There are a few flaws, but they can probably be worked with if the project directors will realize the limitations and plan and schedule the process to take them into account rather than force the methodology to fit into a timetable. This is difficult with a volunteer organization, but critical to achieve the advantages expected from the modeling approach. I hope that it is possible to remain true to the original process, even with the difficulties.

The primary problem at the application committee level is actually getting the members to think "in the information plane". The normal day-to-day work of most committee members is with processes - that is, getting things accomplished; doing things to other things, and with things. We can imagine that these processes occur in a two-dimensional plane; let us call it the 'process plane'. The 'information plane' is where information models reside, and is orthogonal to the 'process information plane', and KEEPING THEM THERE is the real problem. Countless hours have been spent discussing important issues (in the field being modeled) that are irrelevant to the way that information about that field is related. The problem of not understanding the difference between the process of doing FEM and the list of entities required to carry out that process, and their interrelationships keeps coming up. If it seems that "important issues" shouldn't be "irrelevant", remember that the model is of data and the relationships between data, and NOT of the process of finite element modeling and analysis.

9

It is critically important that at least one person with a proper grasp of information modeling attend the majority (all is better) of committee meetings when a model is being formulated. This is especially important with the long hiatus between meetings, and the variable membership of the committees. Considerable time was spent each time attempting to bring new members up to speed with IDEF, and keeping forgetful old members in the information plane. If no reliable and respected modeling semi-expert is available, much time will be wasted producing a model of a process, not an information model. This is almost guaranteed.

The second potential problem seen with the methodology is the error of trying to run all three layers simultaneously. Given the time constraints and the exploratory nature of the PDES Initiation Effort, it was acceptable and probably necessary to run all three processes simultaneously. However, I would object strongly to running the full scale PDES project on the same schedule. If a technically solid PDES with good long term viability is to be developed, adherence to the methodical and systematic approach offered by the three schema architecture is important.

The third problem with PDES is conceptual. While I support the concepts and goals expressed by many of the people contributing to this effort, I strongly suspect that not one in five of the committee members has a real feel for the size of the task being undertaken. There has probably been no previous attempt to organize the world over such a broad range of applications. I'm fairly sure that the project is much larger than most individual applications experts understand, and that without proper understanding of the scope of this project it may flounder or even fail. If the project is intentionally limited in scope it could succeed. If the PDES project is kept as is defined (rather open-endedly), AND SCHEDULING IS OVERLY OPTIMISTIC, the appearance of failure will be the result of regardless of the quality of the work. The scope of the project is great, and realistic scheduling is needed, rather than limiting the scope.
The need for high level planning on scope is in addition to the need for a better understanding of the FUNDAMENTAL interrelationships between different technology areas encompassed by the PDES project. This need was first (and far more competently) addressed at the Knoxville IGES meeting by Roger Gale, and I recommend that his counsel on this subject be sought.

On the plus side, the natural language quality assurance loop was used successfully to verify both the veracity of the modeling for FEM, and to validate the conversion from IDEF to IA modeling language. This is a valuable tool, and should be used in the future.

10

Liaison between the application committees and the logical layer integration personnel is very important. I frequently interacted with John Zimmerman during the Initiation Effort, and it is amazing how severe the "can't see the forest for the trees" problem can be. Application committee members frequently feel that a model is clear when it is not, and logical layer integration has a tendency to inadvertantly modify or even destroy relationships. Liaison between these areas should be close and occur frequently.

Overall, the PDES modeling approach is a real advancement over the previous haphazard, cut and past approach to a standard. A good bit of project leadership and committment by qualified information modeling experts is required to make it work. I sincerely hope that the project is successful since it will be a real landmark of integration, communication, and order in a disorderly world.

Appendix B

Original Logical Layer Methodology Paper

PDES Initiation Logical Layer Methodology

Principal Author: John Zimmerman, Allied/Bendix Aerospace, Kansas City

The primary mission of the Logical Layer Initiation Effort is to develop the definition of the logical entities required to meet the needs of all applications within the initiation scope. This includes "generic entities" which are required by two or more applications and "application specific entities" which are used in only one application area. This set of entities and their relationships is referred to as the conceptual schema.

The Primary goal of the PDES Initiation Logical Layer Methodology is to guide the Logical Layer Initiation Effort in the development of the conceptual schema for Task 1 and Task 2 of the PDES Initiation Effort as described in the PDES Logical Layer Charter.

The following are key success factors for this methodology:

1. Maximize usage of state-of-the-art conceptual modelling techniques and tools.

2. Support the integration of the three PDES architectural layers (application, logical, and physical) as described in "The Second PDES Report".

3. Maximize the usage of human resources in development of the conceptual schema by defining and establishing project roles.

4. Simplify the methodology as much as possible.

5. Give the methodology growth potential so that it can support future PDES efforts.

6. Maximize the potential of the conceptual schema to serve as a central resource from which all other PDES forms can be computed.

Methodology Overview

Refer to the accompanying figure for a graphical overview of the methodology. The numbering of the models is consistent with the US Position paper "Reference Models, Development Methodology, and Entity Subsets for STEP" submitted to ISO/TC184/SC4/WG1 by the US TAG.

The methodology is broken into three phases:

PHASE 1: Pre-conceptualization. In this phase all application area reference models, regardless of modelling form, are reduced to binary form to maximize potential for conceptualization and integration. When these binary models are verified against the original application area -model they are considered "qualified".

Application

Discipline 1	Discipline 2	Discipline N
2 Application Area Reference Model	2 Application Area Reference Model	2 Application Area Reference Model

Verify   Reduce   Verify   Reduce   Verify   Reduce

Logical

Phase 1
Reduce to binary and qualify

3 Qualified Application Model-Binary	3 Qualified Application Model-Binary	3 Qualified Application Model-Binary

Phase 2
A. Build integrated binary model

B. Build conceptual architecture

4 Conceptual Schema Binary (NIAM)

Verify Mapping

Verify

Phase 3
Group integrated model into a structural model (DSL)

4A Conceptual Schema Structural (DSL)

Physical

5 Exchange Format

7 Product Data

FIGURE 1   5

PHASE 2: Conceptualization and Integration. In this phase conceptual entities and relationships are discovered. An integrated conceptual model is built from which can be derived (via mappings) any application area reference model. A conceptual architecture will be built that models all conceptual categories. This architecture will be stored in a dictionary which will be developed in this stage. It is in this phase that the maximum potential of the conceptual schema will be realized.

PHASE 3: Post-Conceptualization. In this phase the binary conceptual schema will be conditioned in preparation for development of the physical file format. The conceptual schema is grouped into a nested record form called Data Specification Language (DSL). This DSL form is the ultimate deliverable to the physical file layer. The ultimate source of the DSL will always be the binary conceptual schema.

Methodology Details

PHASE 1: Pre-conceptualization

Goal: To maximize the conceptualization and integration potential of the application area models.

Input: Application area reference models.

Deliverable: A computerized binary model verified to be equivalent to the original application area reference model.

Description: This phase of the methodology conditions the application area reference model in preparation for conceptualization and integration. In the PDES Initiation Effort application area reference models may be of any form that has a reasonable degree of rigor. The conditioning process converts these diverse forms into a single standard semantic form. The standard semantic form is the binary model. Specifically, Nijssen's Information Analysis Model (NIAM) will be used.

All application area reference models are manually reduced to binary form and entered into an electronic data base (CDC's Information Analysis Support Tool - IAST). This model database is echoed back to the application area in the form of natural language sentences derived manually from the binary model and in computer-generated relational structures. The relational structures are used for the validation of binary models that have been translated from record-oriented application area models. The natural language sentences are returned to the application area for approval. The binary model is considered to be "qualified" when the natural language sentences are approved and the relational structures are verified to be equivalent to the original record-oriented model. It is presumed that most application area reference models will be record or relationally oriented.

Even though the computer generated relational model may be useful in follow-on PDES efforts, its only use in the Initiation Effort is the validation of record-oriented reference models. The deliverable of this phase is the binary conceptual model.

PHASE 1 roles identified:

1. Model translators

2. Model software support tool technician

3. Liasion between application and logical layer to support natural language verification.

4. Data modelling expert who is able to verify equivalency of the computed form with the original application area reference models.

PHASE 2: Conceptualization and Integration

Goal: To build the conceptual schema and maximize its potential as a central resource in the PDES Initiation Effort and to develop a conceptual dictionary tool to hold t'.e components of a conceptual architecture.

Input: Qualified binary models of the application areas.

Deliverable: A computerized conceptual schema in binary form, and a computerized data dictionary containing the conceptual architecture and conceptual-to-application area mappings.

Description: It is in this phase that the actual conceptualization and integration occurs. The reader is encouraged to refer to Appendix A which is an excerpt from "The Second PDES Report". It overviews and describes the three layer architecture. This phase is the most challenging of the three phases of the logical layer methodology. It is from this phase that the crucial aspects of extensibility, stability, resilience, and technology independence will be confronted. Some cultural resistance to this phase is to be expected as it seems to draw out the process of getting to the ultimate PDES deliverable, the physical file format.

As an initiating effort, this phase will start with popular concepts and notions of conceptual schema that have been derived from ANSI/X3/SPARC, ISO/TC97/SC5/WG3, NASA IPAD, and PDDI. The logical layer team realizes that conceptualization across the broad spectrum of product data represented in the application models is a new area for standards work. The team also realizes that the conceptualization of engineering and manufacturing artifacts is fairly new. The team expects to adapt the principles of conceptualization as they apply to business systems to engineering and manufacturing artifacts as much as possible, but realizes that new conceptualization principles must be developed.

PHASE 2 is divided into two sub-phases, PHASE 2A and PHASE 2B. PHASE 2A is concerned with the building of the conceptual schema from detailed application area reference models (in a general sense a bottom-up process). PHASE 2B deals with a conceptual schema architecture that will give high-level structure for the guidance of PHASE 2A activities (in a general sense a top-down process). It is anticipated that the software support tools for PHASE 2A and 2B will be separate but manually coordinated. The primary support tool for PHASE 2A will be IAST. The primary support tool for PHASE 2B may possibly be a relational data base system such as RIM.

PHASE 2A Description: The construction of a conceptual schema is not a well-defined process and this methodology will serve only as a guide.

The following tasks are identified:

1. Develop a set of entity categories. To the greatest extent possible these categories will be generic in that they may be used across a broad set of applications. These categories can be discovered as the qualified application reference models are being reviewed (bottom-up) or an initial set can be adopted provisionally from other sources such as PDDI.

2. Develop a set of structural categories. A structure in this case is a recognizable pattern of inter-related entities that appears in multiple application area models. The conceptual schema will consist of generic entities, generic structures, and application specific structures.

3. Examine each qualified application area model and attempt to break it completely into generic entities, generic structures, and application specific entities. This partitioning of the application area model is recorded in the data dictionary (this tool is created by PHASE 2B activities.).

4. Once the application area model has been completely translated into the conceptual schema, logical layer workers in conjunction with application layer workers verify that the original application area model can be recovered. It is the job of the logical layer worker to be familiar with the generic entitiies and structures. It is the job of the application area worker to find a best fit for his application entities. A cooperation effort between logical and application workers is crucial in this task.

5. Once the application area model is verified to be recoverable from the conceptual schema all mappings must be defined and recorded in the dictionary. The logical layer team may possibly not be concerned in the Initiation Effort about formally defining these mappings although in future PDES efforts it will become more important. In this case narrative sentences may suffice.

6. Make any adjustments to the conceptual schema (hopefully an addition of a new generic entity, not a change to an existing generic entity). It may be necessary to regressively test all previous application area models for recoverability after a major change to the conceptual schema.

PHASE 2B Description: This activity is similar to the strategic data planning activity for the development of a large integrated business information system. The tasks are as follows:

1. Adopt a conceptual schema architecture. Appendix B is an excerpt from the US position paper "Reference Models, Development Methodology, and Entity Subsets for STEP" submitted to ISO/TC184/SC4/WG. The subsets referred to in this paper are the major architectural components we are looking for. These architectural components will be dictionary categories. Refer to Appendix B for a complete rationalization for the need of a conceptual schema architecture.

2. Develop a conceptual dictionary tool to hold the architectural components. It would be advantageous if this dictionary were an integral part of the IAST but the PDES Initiation Effort delivery schedules will not permit this. It is suggested that a simple relational tool such as RIM be used provisionally. The following dictionary categories would be a start: Life Cycle Stage, Discipline, Functional Area, Application Area, Class, Entity, Version. This initial set of categories should greatly assist in giving the conceptual schema development project direction and cohesiveness. An effort should be made to keep the number of dictionary categories as small as possible. Obviously new categories must be added to cover the mapping of the conceptual schema to application area models.

PHASE 2 roles identified:

1. Model software support tool technician

2. Liason between application layer and logical layer

3. Dictionary administrator

4. Conceptual model administrator

5. Data architect who has broad knowledge of engineering and manufacturing life cycle, disciplines, and applications.

6. Data analyst

PHASE 3: Post-Conceptualization

Goal: To condition the conceptual schema in preparation for conversion to the PDES exchange format.

Input: The binary conceptual schema and mappings to application area models.

Deliverable: A complete specification of the conceptual schema in Data Specification Language (DSL).

PHASE 3 Description: The purpose of this phase is to convert the binary conceptual schema into a grouped logical record form (a structural form). This form is still neutral as it has not yet been committed to a physical form. The Data Specification Language (DSL) has been chosen as the PDES

Initiation Effort standard structural form. It was chosen because its form (nested array) naturally models the hierarchical structure of most engineering and maunfacturing artifacts. It is also a compact textual form suitable for documentation.

The major task is the actual conversion of the binary conceptual schema into DSL. For the PDES Initiation Effort this will be done manually but it is suggested for follow on efforts that this conversion be automated.

The DSL specification of the conceptual schema represents the official documentation, however the binary conceptual schema will always be the source from which the DSL specification is generated. The binary conceptual schema is the principle resource for the Initiation Effort.

PHASE 3 role identified:

1. Translator from binary conceptual schema to DSL

Appendix A

4.4 THE ARCHITECTURE OF THE PDES

The task of developing a data standard for industries using CAD/CAM is a huge undertaking. The problem must be broken down into smaller pieces in order to make progress. Modularization is the separation of a process into small, separate functions with precise interfaces between the functions. This separation makes each function manageable and eases maintenance since changes can be isolated to a specific function. Modularization provides flexibility to make changes since, as long as the interface between functions remains the same, one function cannot tell if another has been changed or even replaced. Modularization of the development process is a philosophy recommended for the PDES development. That is, different groups will be tasked with different functions of the design process. The function of data analysis and design will be partitioned into three parts based on the level of abstraction of the data. Thus, the application information will be separated from the conceptual entity definition which will be separated from the physical definition. Each such partition will be called a layer.

As alluded to above, the PDES will be structured into a three layer architecture. These layers correspond almost identically with the layers of schema in the ANSI/X3/SPARC three schema architecture (cf. appendix C.1.1) for defining and implementing data bases. The layers are described in the following sections and illustrated by figure 4-3.

4.4.1 Application/User Layer

In the chosen architecture, the top layer is the user or application layer. This is the layer at which the ultimate user lives and thinks. He formulates his data requirements in his own terms stating concisely what he needs. He draws from his own experience and from the established terms, conventions, techniques, and methodologies of his discipline. He isn't concerned with the number of different ways that a single thing, event, or phenomenon can be represented. He isn't concerned with analogous notions used by different applications. For example, he doesn't care-that electrical
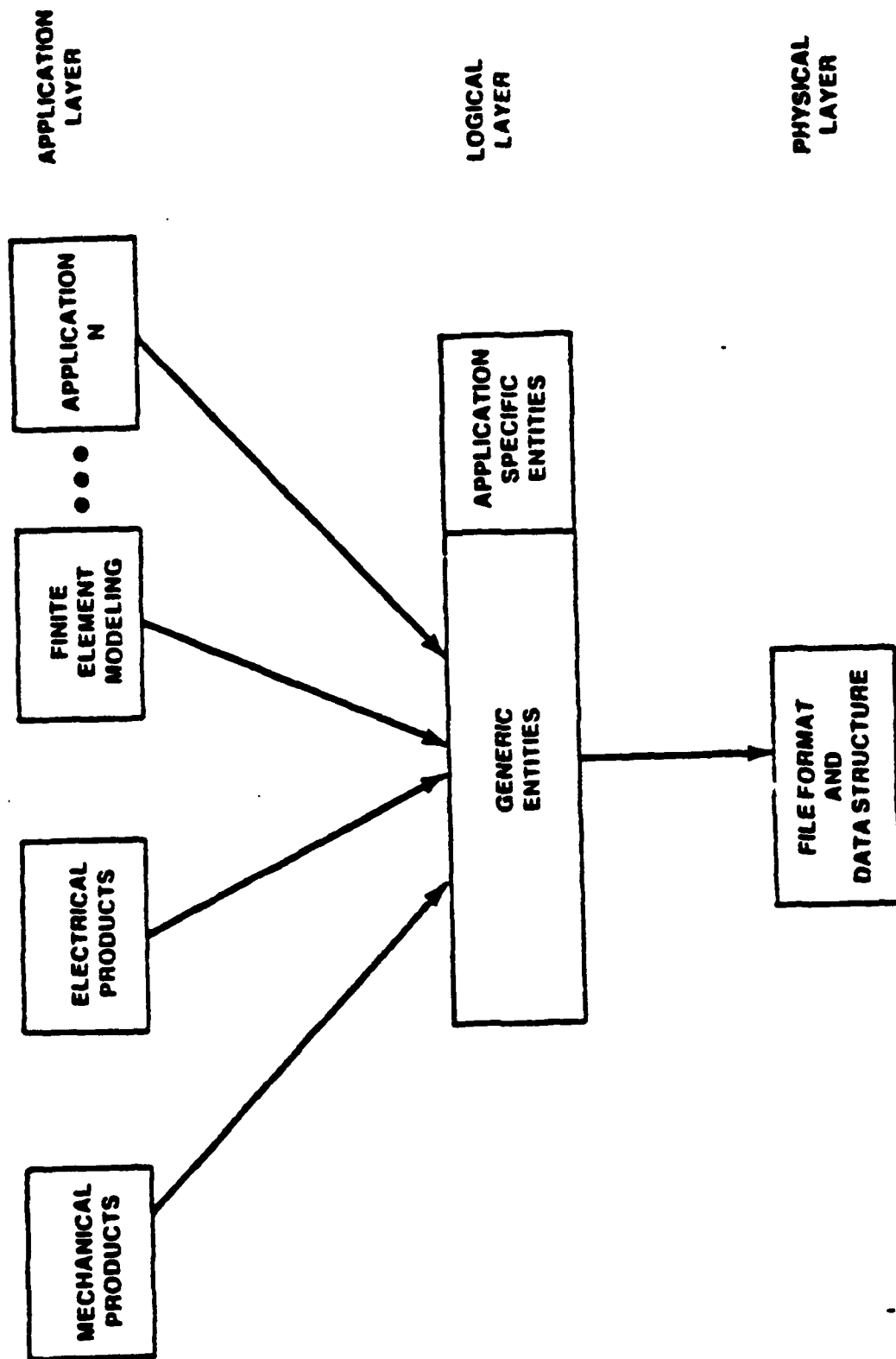
Figure 4-3. Three Layer PDES Architecture

and piping networks share much in common. The different application groups such as Electrical Products or Finite Element Modeling define the information relevant to their application and model the interrelationships that exist between the informational entities. This information is defined by using a reference or information model. The reference model helps structure and validate the data in the application.

This layer of the Standard contains as many different applications and entities within those applications as there is apparent need for.

4.4.2 Logical/Conceptual Layer

The second layer is the logical or conceptual layer. This is where the data content for the set of generic entities is defined. This set should be a normalized, minimumly redundant set (cf. sec. 4.3) that supports the information defined by the applications.

At this layer, logical commonality will be sought across all applications. Things, events, and phenomena which are identical except for the renaming of components will be treated as being logically identical. Thus the connectivity of a piping network and an electrical network will be considered logically identical. Also at this layer, there will be exactly one way, not several, to represent a directed line segment, perhaps by a point together with a direction vector.

At this layer, complex things, events, and phenomena will be constructed of less complex things, events, and phenomena whenever possible. The purpose is to maximize the utilization of conversion processors for simpler entities in the process of converting a complex entity.

Similarities in the information requirements of different applications will be integrated into single conceptual entities. The integration of different application requirements will control the definition of redundant entities and will help to ensure a consistent, coherent entity set.

The entity definitions at this layer will be in a logical form. That is, the data content of an entity will be defined but not the physical format. As described in Section 4.2, a definition language will be defined in which the entities will be specified. This language will be a formal, rigorous language which will help reduce ambiguity as is present in the IGES. In addition, reference models will be built to verify consistency among the definitions.

The data content of the set of application-specific entities will also be defined at this layer. These entities will be defined in the same rigorous manner as the generic entities above, that is with their data content specified using a formal definition language and reference models.

It is expected that the concepts of the logical layer will be organized as the product data itself is organized. This organization is discussed in section 2.1 and is outlined in figure 2-1.

4.4.3   Physical/Internal Layer

The bottom layer is the physical or internal layer. This layer contains one or more actual file format definitions. It consists of the description of the sections, records, fields, sequencing, and associated formats for the exchange file. Again, a formal definition language will be used to reduce ambiguity. A reference model will be built for each format definition.

Since the content is separated from the format, multiple formats can be defined for logical entity definitions which only affect the read/write routines of a processor. Thus the majority of a PDES processor will be independent of the file format.

Appendix B

6.0 Subsets of the Conceptual Schema

One or more mechanisms are required within the Conceptual Schema for defining subsets of entities. These will be used in a variety of ways for creating, understanding and using STEP.

6.1 Requirements for Subsets

6.1.1 Human Understanding of the Standard

o To understand any large collection of facts (such as the Conceptual Schema), a human must categorize the facts into collections (subsets) which are intellectually manageable.

-11-

o The names on the subsets can be used to understand the scope of the collection.

o Subsets for the Conceptual Schema act as a directory for users or logical modelers (developers) to find existing entities that perform a function.

o An entity can be better understood if other members of its subset and the nature of the subset are known.

o Users can match an application with others which can do the same or related jobs based upon Conceptual Schema subsets.

6.1.2 <u>Functional Requirements</u>

o Subsets provide an abbreviation efficiency within the reference models. For example, when a certain attribute may be any valid curve, we can specify the class "CURVE" rather than enumerating all valid curves.

o Schema management procedures may be used to propagate common attributes to every member of a class.

o Subsets can be used by validation checking software to certify translators and other applications.

6.1.3 <u>Management of Development</u>

o Subsets of the Conceptual Schema represent subsets of the work required in developing the standard. They can be used to define the scope, development milestones, and subdivision of labor and expertise.

o A uniform system of subsets may be useful in recognizing voids in the standard. For example, if analysis reference models had been defined for mechanical and architectural disciplines, but not for electrical, the void would be obvious.

o Versions of the standard could be regarded as natural,
  time-dependent, subsets.

6.2 Identification of Subsets

The best method of subset identification would be to exercise the
methodology discussed in Section 4 and use logical layer processing to help
discover natural functional subsets. However, logical layer processing
would be enhanced by a pre-existing, coordinated set of subsets selected
from common knowledge of data, functions, and applications within the
CAD/CAM community. This set will undergo continual review and updating as
the standard develops in the logical layer deliberation.

6.3 Proposed Subset Types of the Conceptual Schema

To address all of the above, a network of subsets of several types are
required. The structural relationships between the types is depicted in
Figure 3. The types are defined as follows:

Versions - Time sequenced sets of the entire standard. Each
           version would contain all entities and subset
           structures valid in a particular release of the
           standard.

Functional Area - A high level set of application area subsets can be
           used for a particular function. It can be regarded as
           a two dimensional matrix of engineering disciplines and
           product life cycle as shown in Figure 4. Each cell on
           the matrix defines a functional area and may contain
           multiple application area subsets. This chart and
           concept is adapted from document (4). In that document
           the matrix is part of a 4 dimensional matrix called
           "Level". The term "Level" derived primarily from
           another dimension which classified geometric
           complexity. That concept is part of a following-class
           structure.

-13-

Subset Types in the Conceptual Schema
As Related to Entities
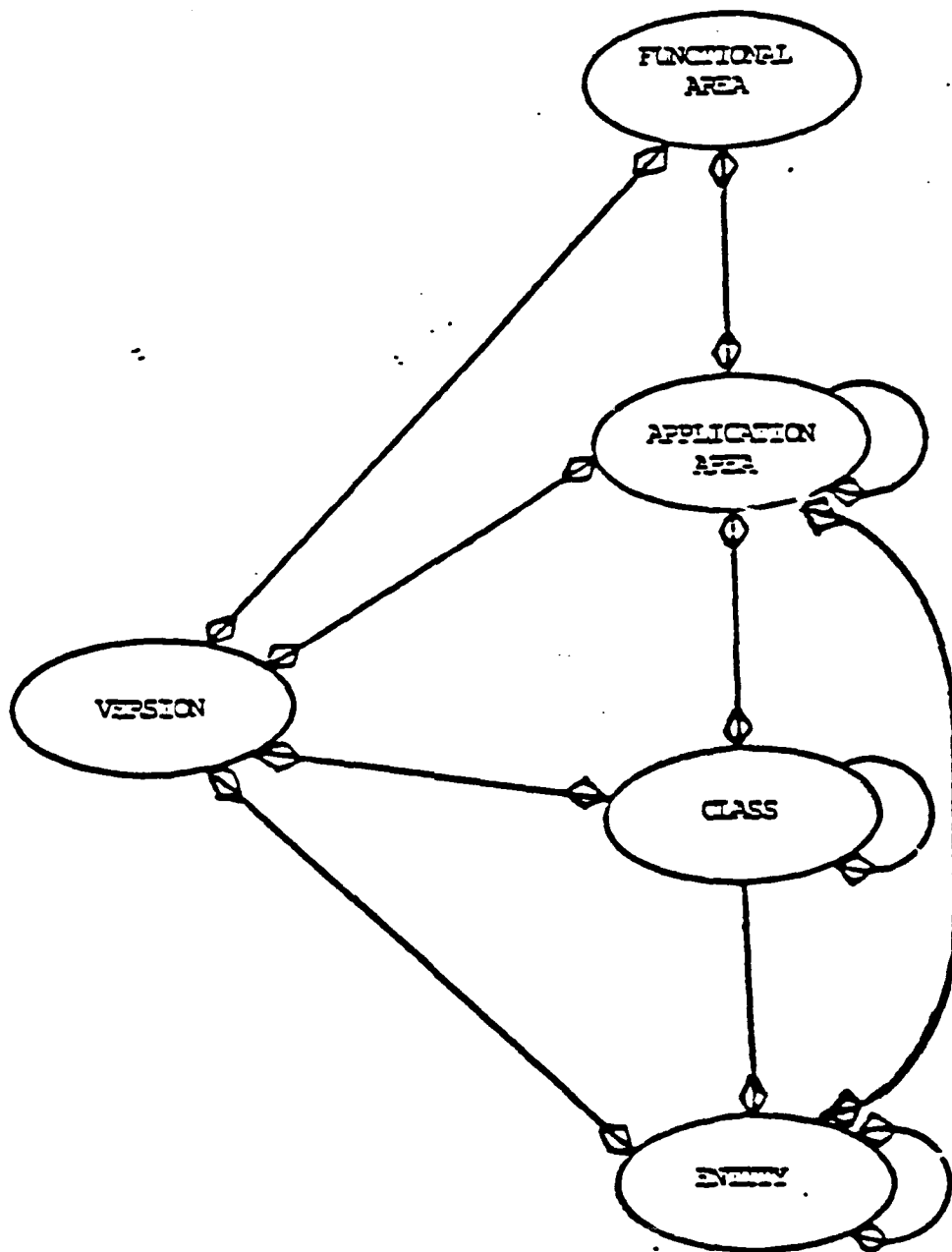
FIGURE 3

FUNCTIONAL AREAS

| DISCIPLINE | PRODUCT LIFE CYCLE | | | | | |
	DESIGN	ANALYSIS	PRODUCTION	ASSEMBLY	QUALITY CONTROL	PRODUCT SUPPORT
MECH. ENG.						
ELECTR. ENG. ELECTRONICS						
ARCHIT., CIVIL ENG.						
PLANT DESIGN						
GEODESY, CARTOGR.						

APPLICATION AREAS

o FEM
o HEAT TRANSFER ANAL.
o KINEMATIC ANAL.
o FIT ANALYSIS
o RADAR SIGNATURE
o MACH. PART DESIGN

FIGURE 4

-15-

Application Area - A set of entities and classes for modeling the concepts of a particular application (such as forging design) or related applications (such as forging design and manufacture). Application area subsets may apply to multiple Functional Areas; e.g. FEM Application Area will be used for analysis in several disciplines. An application area may contain other application areas; e.g., manufacturing forging area may be different from but may include design forging area. A combination of version and application area could be used to specify capability of an application.

Class - A set of entities or classes (but not both) which are semanticly similar. Each entity is contained in exactly one class. Each class is contained in zero or one higher level class.

A class may be generic or application area specific. This means it may be used in multiple application areas or may simply collect the entities which are unique to a single application area.

In Figure 3 the relations between subset types and entities are represented as single diamond leaders for one-to-many relations and double diamond leaders for many-to-many relations.

See Appendix B for the recommended classes and entities for STEP.

6.4 Storing Subset Definitions

One method for defining subsets within the conceptual schema is the "class" structure defined in the Data Specification Language in WG1 N20. Additional methods may be required, particularly for versions.

STEP Classes

2.1.1 Geometry/Topology
2.1.2 Tolerance
2.1.3 Form Feature
2.1.4 Part/Assembly
2.1.5 Administrative & Control   Generic Design Classes
2.1.6 Constraint Dependency   (Used by Multiple Life
2.1.7 Material   Cycle Stages)
2.1.8 Process
2.1.9 Instances

2.2.1 Analysis
2.2.2 Manufacturing
        o Planning
        o Fabrication   Classes for Specific
        o Assembly   Stages of Product Life
2.2.3 Quality Assurance   Cycles OR Specific
2.2.4 Testing   Application Areas
2.2.5 Product Support

2.3.1 Product Manifestation
        o Documentation
            o Drafting
            o Bill of Materials
        o
        o Display
2.3.2 Metadata
2.3.3 Parametric Design
2.3.4 Data Base
2.3.5 User Defined Entities

Appendix B

Figure 1

Appendix C

Logical Layer Content

Task 1 Deliverables

    C1 Wireframe Geometry Resource Model

    C2 Presentation Resource Model

    C3 Flat Plate Mechanical Part Discipline Model

Task 2 Deliverables

    C4 Electrical Schematic Application Model

    C5 Tolerancing Application Model

    C6 Finite Element Modeling Application Model

    C7 Topology Resource Model

    C8 Geometry-Topology Associativity Resource Model

Appendix C1

Task 1

<u>Wireframe Geometry Resource Model</u>

(RESOURCE)

C1.1   Initial Model and Documentation

C1.2   NIAM Model

C1.3   DSL Model

Appendix C1.1

Task 1

Wireframe Geometry Initial Model and Documentation

(RESOURCE)

PROPOSED WIREFRAME GEOMETRY FOR PDES

Edward Clapp
IBM

The first draft of this document was the result of the February 1985 Cincinnati meeting of the PDES Logical Layer Initiation Task Group. The major influences for this proposal were IGES 3.0, the IGES Experimental Solids Proposal, and the personal idiosyncrasies of those present at that meeting.

I have received written comments from a number of people, which are available as Curves and Surfaces Committee documents CS85-2 to CS85-10. We have also discussed it at two IGES meetings. Unfortunately it has not been possible for me to present it to the ISO TC184 SC4 WG1, though the German delegation has extensively commented on it. All of these have been most welcome. My summary of these comments and responses to them is document CS85-16.

This document is the proposed wireframe geometry for PDES. There are changes but this does not represent a major technical revision of the first draft in terms of the content of the geometry. I have attempted to more fully explain some of the constructs and the reasons for them. Where there was controversy, I have so indicated.

The entity set has been divided into two parts. The first is that which is explicitly geometry. The other is a collection of entities needed to support geometry and a discussion about the things that we need to be able to say symbolically about the geometry. This includes some discussion on definitions and instancing, external referencing, and grouping.

The text entity has been dropped as the Presentation portion of PDES is now well underway.

I think this paper does address the wireframe portion of geometry adequately. It is not complete in that:

- we need some sort of formal language to express these constructs precisely. I have used an informal Pascal-like language here. For the geometry of curves, this is sufficient to understand the intent. For the kinds of ways we need to be able to relate the geometric entities, it is not.
- parameterization of curves, needed to define surfaces, is incomplete. This will be addressed when we start work on surfaces.
- 2D offset curves can be added when parameterization has been completed.
- some mechanism for parametric evaluation of geometry is needed, but that is beyond the scope of this effort.
- geometric tolerancing is inadequate. Contributions from anyone wishing to make them are welcome.

- topology for B-rep modeling needs to be addressed in the context of this proposal.
- also needed is a general framework in which these entities are used; in particular descriptions of some of the structural entities such as model, view, drawing, and library.

The people who helped most in preparing the original document were:
    Steve Bodnar, CDC
    Richar Fuhr, Boeing
    Don Hemmelgarn, ITI
    Tom Railing, ITI
    Doug Schenck, McDonnell Douglas
    Chia Hui Shih, SDRC
    Peter Wilson, GE

Subsequently, J.C. Kelly (Sandia), John Zimmerman (Bendix), and Paul Thompson (CDC) made a number of helpful comments as a result of attempting to use the document in creating an IA model for wire-frame geometry.

However, final responsibility for any errors and/or omissions is mine. Comments, verbal or in writing, are requested and should be directed to me at

    IBM Dept 75E
    11601 Wilshire Blvd.
    Los Angeles, CA 90025

    Telephone: (213) 312-5975.

The primitives are positions, directions, and scalars. The geometry is limited to points and curves, both bou.ded and unbounded. Vectors are defined but not used, as we found it more natural to work with the direction portion in constructing the conics.

All entities may be named; however the entity name is always optional. The symbolic use of names has not been completely worked out yet. For this document the type 'Pointer' will be used to hide some of the various ways entities may be used or referred to. This is a complex problem that is not fully addressed here. The examples at the end of this section and the section on grouping mechanisms provide some discussion of this issue.

We came to a number of conclusions about the geometry and how it should be treated:

We attempted to pick a minimal set of geometric entities. Where this rule has been violated, I have explained the reasoning behind our choices.

2D and 3D geometry have been separated explicitly. This saves space for 2D data and makes it explicit when data is planar. This has been a controversial issue. However, I think that we made the correct choice. There are systems that are inherently 2D: drafting coordinates, parameter space, and 2D viewing coordinates. Offset curves need to be 2D. The IGES AEC committee did a survey of vendors' use of subfigures which indicated that the distinction would be of use.

It is necessary to be able to say as much symbolically as is practical. It is better for an exchange file to be able to state that two curves share a common endpoint or that two composite curves share a segment than to try to compute this sort of information. Since we are attempting to build a standard that will be able to grow to meet future needs, these sorts of capabilities need to be built into the standard even if there is no great need for them at present. This is especially true for solid modeling considerations.

A generic set of entities should be used for geometry so as to be able to use them also to deal with topology and numeric control. I did not discuss here how to implement these other uses (but hope to deal with topology no later than 3/86).

We need ultimately to say precisely when an entity is well defined. Some of this is syntactic and some semantic. The formalisms for the syntactic aspects are well known and understood; those for the semantics are not. The geometric tolerancing issues discussed here are the beginnings of that issue as seen by the computational geometer.

We have tried to choose the most stable representations for the commonly used CAD/CAM geometric entities. By stability we mean

that a small change in the data for the entity, due perhaps to
truncation in its ASCII representation, results in a small
change in the pointset defined by the geometry. The case of
the original definition of the IGES conic entity is a good
example of an unstable representation.

PART 1: GEOMETRY

The Wireframe geometric entities are points and curve segments. To aid in their definition we have some auxiliary entity types:

    Positions
    Directions
    Vectors
    Curves
    Transformation matrices

Tolerancing is addressed here as strictly a geometric approximation issue.

Position

```
Position2 = entity_type
        Name : optional string;
        X    : real;
        Y    : real;
        end;

Position3 = entity_type
        Name : optional string;
        X    : real;
        Y    : real;
        Z    : real;
        end;
```

There is a subtle distinction made here between Positions and Points
(defined later). Points are considered to be CAD/CAM entities which
are to be translated as geometry; Positions are constructs used in
defining geometry (for example, control points for a B-spline). In
use in an exchange syntax, one would want to allow any Pointer to
a Position to be satisfied by a Pointer to a Place.

The objection to homogeneous coordinates is that they are only
used for control points for splines and it then becomes difficult
to symbolically express that the same point is being on two or
more curves if the homogeneous coordinate is different.

Direction

_____

_____

```
Direction2 = entity_type
            Name : optional string;
            X    : real;
            Y    : real;
            end;

Direction3 = entity_type
            Name : optional string;
            X    : real;
            Y    : real;
            Z    : real;
            end;
```

where
  $(X^{**}2 + Y^{**}2)^{**}1/2$ $= 1$ for Direction2
  $(X^{**}2 + Y^{**}2 + Z^{**}2)^{**}1/2 = 1$ for Direction3

We have made Position and Direction the lowest level of what can be
addressed symbolically. It might be desirable, and it would be
simple, to allow addressing the component scalars.

Vector

_____

_____

```
Vector2 = entity_type
          Name      : optional string;
          Direction : Position2;
          Magnitude : real;
          end;

Vector3 = entity_type
          Name      : optional string;
          Direction : Position3;
          Magnitude : real;
          end;
```

This representation allows for least error in cases where the
vector magnitude is small. It also makes it easy to symbolically
express that two or more vectors have the same direction.

Transformation Matrix

```
Trans_mtx2 = entity_type
        Name      : optional string;
        NewX      : Pointer(Direction2);
        NewY      : Pointer(Direction2);
        NewO      : Pointer(Position2);
        end;

Trans_mtx3 = entity_type
        Name      : optional string;
        NewX      : Pointer(Direction3);
        NewY      : Pointer(Direction3);
        NewZ      : Pointer(Direction3);
        NewO      : Pointer(Position3);
        end;
```

The transformation matrix is as defined in IGES. Most systems
probably store the data in this fashion. The rotation portion is
viewed as orthonormal vectors. As such, they must meet the criteria
for being perpendicular and unitary established under the heading of
Geometric Tolerance.

If one wishes to scale in any direction, this can be done separately
(consistent with the definition and use of vectors), with scale factors
for the different directions.

Transformation matrices may be applied to geometry, instances of
geometry, and to groupings that have explicitly been labelled as
movable (or copyable) in space.

Curve

A curve is to be thought of as a parameterized, continuous, possibly unbounded curve in space. The curve entities are:
- line
- circle
- ellipse
- parabola
- hyperbola
- spline
- composite curve

Note that the last three of these have bounded parameter spaces.

The choices made for the representations of the conics reflect:
- geometric stability.
- one of the many conflicting sets of design intent.
- the expectation that most systems already have available procedures to go between their native system representations and the geometric ones used here. This is one of the major reasons why the B-spline was not chosen as the sole representation form for curves.

Curves are bounded by the segment entity type. A Segment consists essentially of a Pointer to a curve (or the definition of the curve embedded within the segment) plus the endpoints of the segment.

A number of the comments suggest that the first draft failed to describe our intent as to the use of curves and segments with sufficient clarity. We had intended that the language for PDES should make much of this transparent to systems not interested in the flexibility we created. We do not, in general, expect that people will be passing lines, conics, and splines as they do now in IGES. Rather, they will be passing segments composed of these entities. We have tried to set up the information content of these entities so that the PDES syntax could deal with them in a straightforward and efficient manner. The examples given towards the end of this paper should be of some help in understanding this issue.

Some notation has been set up to describe the parameterization of the conic sections:

Mtx(A,B,C) is the rotation matrix created by using 3D vectors A, B, and C as the columns of the 3X3 array. If A and B are 2D, the third coordinate of the corresponding 3D vector is 0.0.

Tr(X) is the transpose of the vector X.

A X B is the cross product of vectors A and B.

Line
―――

```
Line2 = entity_type
        Name    : optional string;
        Pt2     : Pointer(Position2);
        Vt2     : Pointer(Direction2);
        Parm    : optional parameterization;
        end;

Line3 = entity_type
        Name    : optional string;
        Pt3     : Pointer(Position3);
        Vt3     : Pointer(Direction3);
        Parm    : optional parameterization;
        end;
```

where
   Pt2 (Pt3) is a point on the line.
   Vt2 (Vt3) is the direction of the line.

and the default parameterization is
   $C(u) = Pt + Vt*u$

Two lines are identical as point sets if they have the same named
Position and Direction (i.e., the referencing is symbolic). When
we define parameterization techniques, they may still be different
lines if the parameterizations differ. This distinction between
point sets and parameterized curves will be maintained for the conics.

```
Circle
_____


Circle2 = entity_type
          Name     : optional string;
          Center   : Pointer(Position2);
          Radius   : real;
          Sdir     : optional Pointer(Direction2);
          Parm     : optional parameterization;
          end;

Circle3 = entity_type
          Name     : optional string;
          Axis     : Pointer(Direction3);
          Center   : Pointer(Position3);
          Radius   : real;
          Sdir     : optional Pointer(Direction3);
          Parm     : optional parameterization;
          end;


where
  Axis    is normal to the definition plane of the circle
            (the value of Axis for the 2D case is (0,0,1)).
  Center is the center of the circle.
  Radius is the radius of the circle
  Sdir points from the center to the start point of the circle.

and the default parameterization is
  C(u) = Center
         + Radius*Mtx(Sdir,Axis X Sdir,Axis)*Tr(cos u,sin u,0.0)
```

Ellipse
─────

```
Ellipse2 = entity_type
          Name    : optional string;
          Center  : Pointer(Position2);
          Mjdir   : Pointer(Direction2);
          Axmj    : real;
          Axmn    : real;
          Parm    : optional parameterization;
          end;

Ellipse3 = entity_type
          Name    : optional string;
          Axis    : Pointer(Direction3);
          Center  : Pointer(Position3);
          Mjdir   : Pointer(Direction3);
          Axmj    : real;
          Axmn    : real;
          Parm    : optional parameterization;
          end;
```

where
  Axis   is the normal to the definition plane of the ellipse
          (the value of Axis for the 2D case is (0,0,1)).
  Center is the center of the ellipse.
  Mjdir is the direction of the major axis of the ellipse.
  Axmj   is the magnitude of the major semi-axis.
  Axmn   is the magnitude of the minor semi-axis.

and the default parameterization is
  C(u) = Center
          + Mtx(Mjdir,Axis X Mjdir,Axis)*Tr(Axmj*cos u,Axmn*sin u,0.0)

```
Parabola
_____

Parabola2 = entity_type
         Name    : optional string;
         Vtxpt   : Pointer(Position2);
         Faxis   : Pointer(Direction2);
         Fdist   : real;
         Parm    : optional parameterization;
         end;

Parabola3 = entity_type
         Name    : optional string;
         Axis    : Pointer(Direction3);
         Vtxpt   : Pointer(Position3);
         Faxis   : Pointer(Direction3);
         Fdist   : real;
         Parm    : optional parameterization;
         end;


where
   Axis  is the normal to the definition plane of the parabola
            (the value of Axis for the 2D case is (0,0,1)).
   Vtxpt is the vertex of the parabola.
   Faxis is the direction of the axis of the parabola.
   Fdist is the focal distance of the parabola.

and the default parameterization is .
   C(u) = Vtxpt
          + Mtx(Faxis,Axis X Faxis,Axis)*Tr(u**2/(4*Fdist),u,0.0)
```

Hyperbola
_____

```
Hyperbola2 = entity_type
          Name    : optional string;
          Center  : Pointer(Position2);
          Mjdir   : Pointer(Direction2);
          Axmj    : real;
          Axmn    : real;
          Parm    : optional parameterization;
          end;

Hyperbola3 = entity_type
          Name    : optional string;
          Axis    : Pointer(Direction3);
          Center  : Pointer(Position3);
          Mjdir   : Pointer(Direction3);
          Axmj    : real;
          Axmn    : real;
          Parm    : optional parameterization;
          end;
```

where
  Axis   is the normal to the definition plane of the hyperbola
         (the value of Axis for the 2D case is (0,0,1)).
  Center is the center of the hyperbola.
  Mjdir is the direction of the major axis of the hyperbola.
         When positioned at Center, it points to the chosen branch
         of the hyperbola.
  Axmj   is is the scalar magnitude of the major semi-axis of the
         hyperbola.
  Axmn   is is the scalar magnitude of the minor semi-axis of the
         hyperbola.

and the default parameterization is
  $C(u)$ = Center
         + Mtx(Mjdir,Axis X Mjdir,Axis)*Tr(Axmj*sec u,Axmn*tan u,0.0)

  where $-PI/2 < u < PI/2$.

Spline
―――――

```
Spline = entity_type
        Name        : optional string;
        Dim         : integer value(2..3);
        Order       : integer value(2..maxint);
        Ctlpts      : array(0..K) of Pointer(Position);
        Weights     : optional array(0..K) of real;
        Knts        : case of
                        array(-(Order-1)..(K+1)) of real;
                        array(-(Order-1)..(K+1)) of integer;
        end;
```

where
Dim	is the dimensionality of the spline.
Order	is the order of the polynomial.
Weights	is the array of weights of the control points if the spline is a rational one.
Ctlpts	is the array of control points. They are all either Position2 or Position3 and must agree with Dim.
Knts	is the knot sequence. It is either an array of real numbers or an array of integer values (in the uniform case).
K	is one less than the number of control points in the spline. It is arbitrary but must be greater than 0.
maxint	represents the largest integer the computer can represent. This is merely a notational convenience for saying that the order must be at least 2 - i.e., the spline must be at least linear - but otherwise of arbitrary order.

The B-spline was chosen as it:
- provides a stable representation form;
- it completely describes the class of polynomial splines;
- is used by a large (and increasingly so) percentage of CAD/CAM systems as their representation scheme for curves.

The spline has been limited to being 2 or 3 dimensional. For NC and other use in the future it may be desirable to relax this restriction. This why we did not create types spline2 and spline3.

Further clarifying details of computation of B-splines may be found in the IGES documents. The spline information used here and the notation, except for the uniform spacing case, is as found in the IGES 126 entity.

It is desirable that the syntax of the file structure be able to ex-
press repeated knots and control points in some compact manner. One
way of doing this would be for the default values in sequences of tuples
to be the coordinate of the previous tuple.
Examples:
(1) in a knot sequence,
     2,,3
would be the same as
     2,2,3
(2) in a sequence of ordered pairs
     2,3,,,6,7
would be the same as
     2,3,2,3,6,7
if the sequence were really (2,3),(2,3),(6,7)
(3) in a sequence of ordered pairs
     2,3,,5,6,7
would be the same as
     2,3,2,5,6,7
if the sequence were really (2,3),(2,5),(6,7)

Composite Curve

```
comp_constituent_entity2 = record
                           Constituent : Pointer(Segment2);
                           Sense       : optional boolean;
                           Cntseg      : optional integer values(0..2)
                           end;


comp_constituent_entity3 = record
                           Constituent : Pointer(Segment3);
                           Sense       : optional boolean;
                           Cntseg      : optional integer values(0..2)
                           end;



CCurve2 = entity_type
          Name       : optional string;
          Cntflg     : optional Integer Values(0..2);
          Closeflg   : optional Boolean;
          Seglist    : list of comp_constituent_entity2;
          end;

CCurve3 = entity_type
          Name       : optional string;
          Cntflg     : optional Integer Values(0..2);
          Closeflg   : optional Boolean;
          Seglist    : list of comp_constituent_entity3;
          end;


where
   Sense is 'T' (default) if the underlying curve segment has the same
      sense as the composite curve.  Otherwise, sense is 'F'.
   Cntflg indicates whether the curve is:
      0 - continuous  (C0 is required)
      1 - has continuous tangent direction
      2 - has continuous curvature direction
   Cntseg indicates the same for the point common between this segment
      and the next.
   Closeflg is 'F' (default) if the curve is open, 'T' if closed.
   Seglist is an arbitrarily large list of constituent entities.
```

The constituent curves must have the correct directionality. That is,
the end point of one is the start point of the next. This is some-
thing that can be said symbolically by having the end of one curve
and the start of the next pointed to by name. It can also be tested
for by determining the points are within a tolerance to one another.
The former method is the preferred one.

They must also have the same dimensionality. The composite curve is
either 2D or 3D so all of the constituent elements must be 2D or they
must all be 3D.

There has been a request that the CO requirement be dropped. This would impose no great hardship on PDES processors or the specification. We would need to spell out in the standard when this was required. E.g., loops in topology would need to be CO.

The parameterization of the composite curve is the inherited parameterizations of the constituent curve segments.
That is, if we let

      C   be the composite curve;
      N   be the number of constituent curves;
      CC(i) be the i-th constituent curve ($1 \leq i \leq N$);
      PS(i) be the parametric value of the start of CC(i);
      PE(i) be the parametric value of the end of CC(i);
      T(i) be the sum from j=1 to j=i of (PE(j) - PS(j)),
              where T(0) = 0.0. (T(i) is the accumulated parametric
              length up to the end of the i-th constituent).

   Then
     (1) the parametric values of C range from T(0) to T(N);
     (2) for $0 \leq u \leq T(N)$,
        C(u) = CC(i)(u - T(i-1) + PS(i))
        where i is such that $T(i-1) \leq u \leq T(i)$.

Point
———

```
Point2 = entity_type
         Name       : optional string;
         Position : Position2;
         end;

Point3 = entity_type
         Name       : optional string;
         Position : Position2;
         end;
```

Curve Segment

_____

_____

```
Segment2 = entity_type
          Name    : optional string;
          Crv     : optional Pointer(Curve2);
          Stpt    : Position2;
          Endpt   : Position2;
          Pst     : optional real;
          Pend    : optional real;
          end;


Segment3 = entity_type
          Name    : optional string;
          Crv     : optional Pointer(Curve3);
          Stpt    : Position3;
          Endpt   : Position3;
          Pst     : optional real;
          Pend    : optional real;
          Path    : optional boolean;
          end;
```

where
  Crv   is the curve that is bounded by the endpoints.
         Crv is optional as the default is a line segment,
         whose definition is in terms of its endpoints.
  Stpt   is the start point on the curve.
  Endpt is the end point on the curve.
  Pst   is the parametric start point of the curve.
  Pend   is the parametric end point of the curve.
  Path   is 'T' if the segment is counterclockwise as seen from the
         Axis direction (the value of Axis for the 2D case is
         (0,0,1)). Otherwise it is 'F'. The default is 'T'.
         This is only needed for the closed curves (circle,
         ellipse, and closed composite curve) when there is
         ambiguity as to the path the segment is on.

The endpoints are the primary means of trimming segments. This makes
explicit when continuity is desired. Parametric values were specified
as being optional and were intended to be used only in those cases
where the curve was self-intersecting.

Pst and Pend must be used if the curve has more than one parametric
value for either the start or end point. That is, if there are $t0$
and $t1$ such that
    $Stpt = crv(t0) = crv(t1)$ or $Endpt = crv(t0) = crv(t1)$
Otherwise, they need not be used.

The direction of the curve segment is from the Stpt to Endpt.

-20-

It is desirable that the syntax of the file structure be able to express curve segments which are defined in terms of already bounded curves (e.g., full circles and ellipses, splines, composite curves) and in have the same endpoints in some compact manner.

Geometric Tolerances

There seem to be two distinct sorts of tolerances:

T_V to indicate that two vectors are perpendicular or parallel.

T_P to indicate that two points are identical or that a point
is on a curve.

T_V can be used as follows:

V1 is parallel to V2   iff $(1-T\_V) <$ abs(V1 dot V2) $< (1+T\_V)$

V1 is perpendicular to V2 iff   0 $<=$ abs(V1 dot V2) $< + T\_V$

V1 has magnitude 1   iff $(1-T\_V) <$ abs(V1 dot V1) $< (1+T\_V)$

T_P can be used as follows:

P1 = P2   iff   dist(P1,P2) $< T\_P$

P1 is on C1   iff   dist(P1,C1) $< T\_P$

where P1 and P2 are points and C1 is a curve.

Also, for open bounded curves one can require that
dist(Start_pt,End_pt) $> T\_P$
and for closed bounded curves one can require tha.
dist(Start_pt,End_pt) $< T\_P$

Of course, computationally one would probably wish to use T_P**2
and dist**2.

Discussions in the IGES Curves and Surfaces Committee suggest that
the sending system should document its criteria for establishing tol-
erances. The receiving system should document its criteria for accept-
ing data and its fixups, if any, and it should output some sort of mes-
sage for any entities that are either altered or rejected.

Examples:
_____

A line segment in the WF proposal is a line with endpoints. There is no reason why the line couldn't be defaulted to the nil name so that the result is a line segment defined by two points. Similarly, with the closed conics (circle and ellipse) the file structure might well be set up so that null endpoints indicate that the segment is a full circle or ellipse. Segments composed of splines and composite curves that are not trimmed could use the same mechanism.

The following would all define the same line segment. It illustrates the systematic approach of being able to use either data or pointers to data. This is a generalization of the approach taken by PDDI. In fact, the second and third cases fit that defining methodology.

There is a more subtle issue of the need to be able to reference the same geometry in different ways that is covered later in the discussion of the grouping mechanism.

The notation used is only meant to be suggestive. One would expect the actual physical representation to be far more compact.

```
(1)   LSA = Segment2
            seg_type = Line2
            Position2(X1,Y1)
            Position2(X2,Y2)
            seg2_def_end

(2)   LSA = Segment2
            seg_type = Line2
            PT1 = Position2(X1,Y1)
            PT2 = Position2(X2,Y2)
            seg2_def_end

(3)   PT1 = Position2(X1,Y1)
      PT2 = Position2(X2,Y2)
      LSA = Segment2
            seg_type = Line2
            PT1
            PT2
            seg2_def_end

(4)   PT1 = Position2(X1,Y1)
      PT2 = Position2(X2,Y2)
      LSA = Segment2
            seg_type = Line2
            Line2((XS,YS),(VX,VY))
            PT1
            PT2
            seg2_def_end
```

```
(5)   PT1 = Position2(X1,Y1)
      PT2 = Position2(X2,Y2)
      L1  = Line2((XS,YS),(VX,VY))
      LSA = Segment2
            seg_type = Line2
            L1
            PT1
            PT2
            seg2_def_end

(6)   PT1 = Position2(X1,Y1)
      PT2 = Position2(X2,Y2)
      PTS = Position2(XS,YS)
      VT  = Direction2(VX,VY)
      L1  = Line2(PTS,VT)
      LSA = Segment2
            seg_type = Line2
            L1
            PT1
            PT2
            seg2_def_end
```

Cases (4) to (6) are subject to the constraint that PT1 and PT2
must be within some tolerance of the line defined by the point
(XS,YS) and the vector (VX,VY).

PART 2: ASSOCIATED ENTITES

These consist of

    Definition and Instance entities
    Group
    External Referencing

Instancing is the use of entities (or structures of entities) defined as quasiprimitives, perhaps with arguments. The term quasi-primitive is deliberately left undefined, but it is intended here to refer to geometry that is use as a primitive definition by the CAD system. Many systems use this technique, along with libraries to contain definitions for use throughout the installation.

```
definition_entity = entity_type
                Name                : string;
                Dim                 : optional integer;
                constituent_list    : list of Pointer(Geometry)
                end;
```

where
   Name is the required name of the definition to be referenced when
        used.
   Dim is the dimensionality of the definition. We are currently
        restricting it to 2 or 3. The default is 3.
   Pointer(Geometry) means that the entities in the list must be
        geometric entities or named instances of them. Nesting of
        instances of other definitions within a definition is allowed
        provided there is no self-referencing.

```
instance_entity = entity_type
                Name        : optional string;
                Def_name    : string;
                trans       : Pointer(trans_mtx);
                def_ref     : Pointer(definition_entity);
                scale       : optional case of
                                 array(1..3) of real;
                                 optional real;
                end;
```

where
   Def_name is the name of the definition being instanced.
   trans   is a transformation matrix used to position the instance.
   def_ref is the name of the definition being used.
   scale   is an optional scale factor for the X, Y, and Z directions
            respectively of the definition entity before it is
            positioned by the transformation matrix. If only one
            value is used (probably the most commonly used mechanism),
            it applies to all three directions.

There has been a request to put a nesting level into the definition. We did not do so because a strict define-before-use semantics would make this unnecessary.

Grouping Mechanism

```
group_entity = entity_type
            Name                : optional string;
            Movable_flag        : optional boolean;
            trans               : optional Pointer(trans_mtx);
            constituent_list    : list of Pointer(Geometry);
            end;
```

where
   movable_flag indicates whether the group is to be considered as a
           physical thing that can be moved in space. The default
           is 'F'.
   trans   is a transformation matrix used to position the group.
           It may only be used if the Movable_flag is set to 'T'.

The movable_flag is optional and its presence indicates that the
geometry defined or instanced in the group is to be moved by any
transformation matrix applied to the group. Note that in case (2)
below the geometry has been defined elsewhere.

No doubt there will be other attributes one might wish to add to
the grouping mechanism, but this seems sufficient for a start.

The issue of the kinds of addressing that may be done is complex and has not really been resolved within IGES. Here is a discussion of some of the kinds of addressing that need to be done.

The elements in the list may be:
(1) defined within the scope of the list.
    ex:
        begin group
            begin point (optionally named)
            .
            .
            end point
            .
            .
            .
        end group
(2) referenced by name from outside the group.
    The geometry already exists within the model.
    ex:
        begin group
            ref PT_A
            .
            .
            .
        end group
(3) an instance of a definition made from outside the group.
    ex:
        begin group
            inst PT_A   •
            .
            .
            .
        end group

(4) an instance of geometry defined elsewhere that already exists
    in the model and is not a definition.

    One example is the same as (3) above. A more illustrative
    example is the following:

        Consider two ruled surfaces:
        RS1 is defined by segments Sa and Sb.
        Sa and Sb are defined inside the scope of RS1.
        RS2 is defined by segments Sb and Sc.
        Sc is defined inside the scope of RS2.

        Suppose one wishes to apply a transformation matrix to RS1.
        There are two choices as to what can happen to RS2. It can
        change, being dragged along by RS1, or it can remain the same,
        being uncoupled from RS1. In the latter case, one might well
        wish to continue to say symbolically the following:

            if the two surfaces were joined together, they would be
            C0 continuous.

If a reference to Sb were used, the first meaning would be
used; if an instance of Sb were used, the second would.

Another way to look at this is to go back to the example in which
a line segment was defined in 6 different ways. Assume for this dis-
cussion that the segment has a transformation matrix associated with
it. Then in cases (3) to (6) one needs to be able to specify that
the transformation matrix is (inst case) or is not (ref case) applied
to the position of the point or line that has been defined outside of
the scope of the segment. The language must be able to express both of
these cases.

External Referencing

```
Xref = entity_type
        Entity_name  : string;
        Entity_class : entity_type;
        File         : optional string;
        end;

Xreftab = entity_type
        Table_name : optional string;
        Refs       : list of Xref_type;
        end;


where
   Entity_name  is the name of the entity in its defining file.
   Entity_class is from (Curve2, Curve3, Segment2, Segment3,
                         Definition_entity, Group).
   File         is the name of the PDES file in which the entity
                is defined.
   Table_name   is the name of the table of external references.
   Refs         is a list of external references.
```

Use of external referencing is dependant upon the general organization
of the PDES file structure, so further details will not be given here.

Appendix C1.2

Task 1

<u>Wireframe Geometry Model (NIAM)</u>

(RESOURCE)

6

WIRE FRAME ENTITY

Sheet 2 of 17

DIRECTION

PLACE

CARTESIAN
COORD.

Direction3

Place3

Cartesian
Coord

(Cartesian)
Value — Real
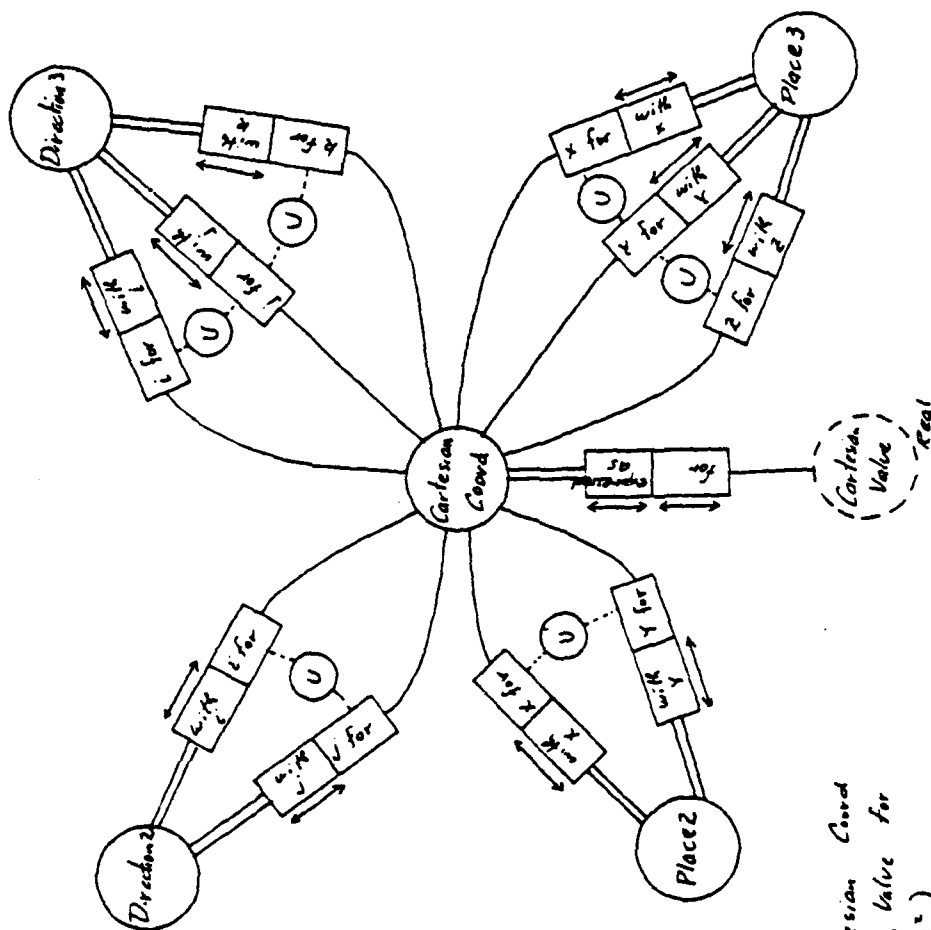
Direction2

Place2

Constraint "unit vectors"
for each d : Direction2
SQRT ((Cartesian Value for Cartesian Coord
i for d)² + (Cartesian Value for
Cartesian Coord j for d)²)

= 10 ±ε

holds

Similar constraint for Direction3.

POINT

VECTOR

DEGREE

SEQUENCE

LINE

Line Parameterization 3

Direction3

Places

Lines

Line Parameterization 2

Directions

Place2

Line2
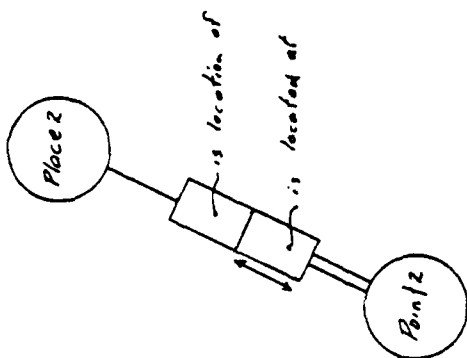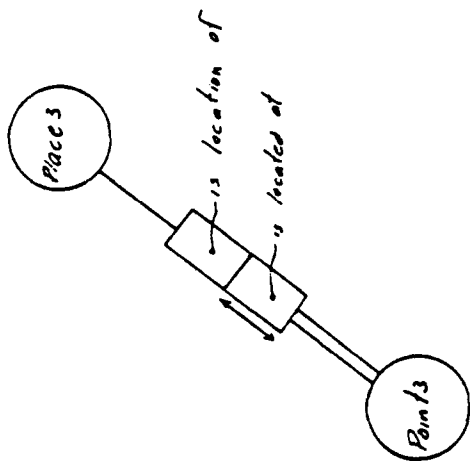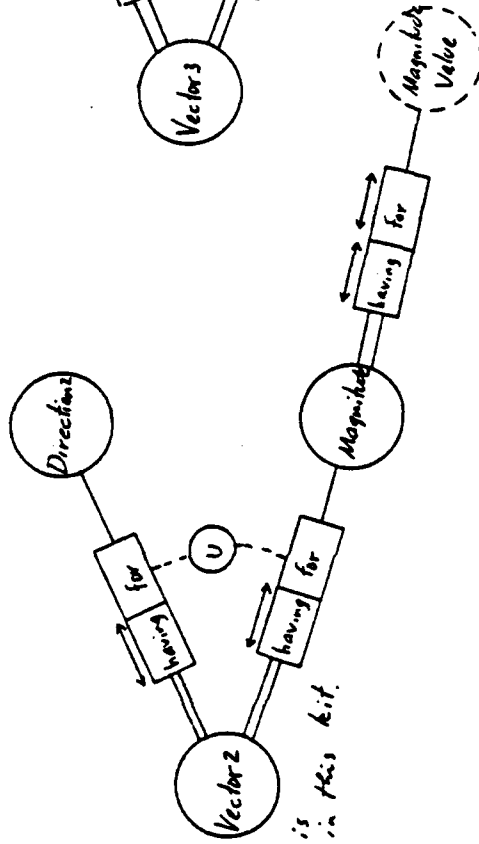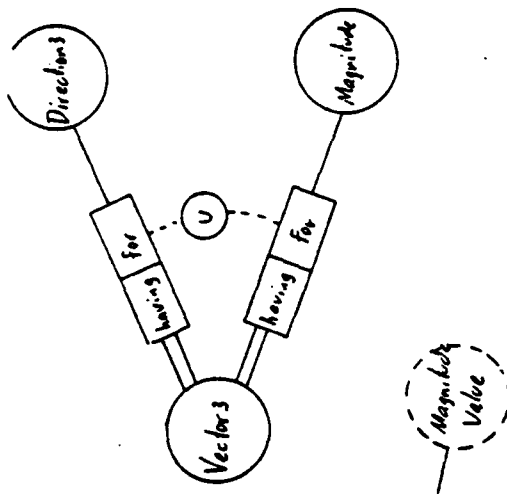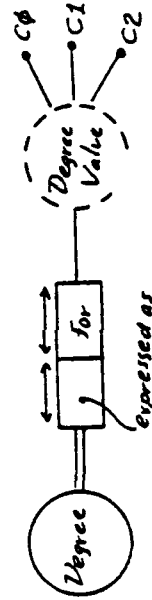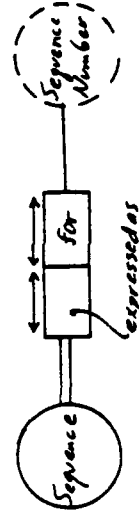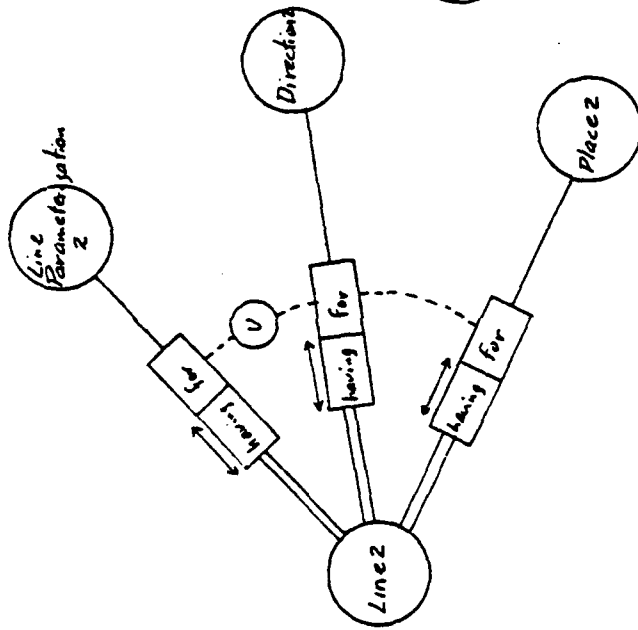
Note: Direction and place do not
uniquely determine a line
because 1) parameterizations
can be different 2) two lines
can share the same point.

CIRCLE

Circle Parameterisation 3

Place 3

Magnitude

Direction 3

Circle 3

having of

Place of — with center of

radius of — with radius of

with start of start of

has axis with — of axis of

Circle Parameters 2

Place 2

Magnitude

Direction 2

Circle 2

having of

Place of — with center of

radius of — with radius of

with start of start of
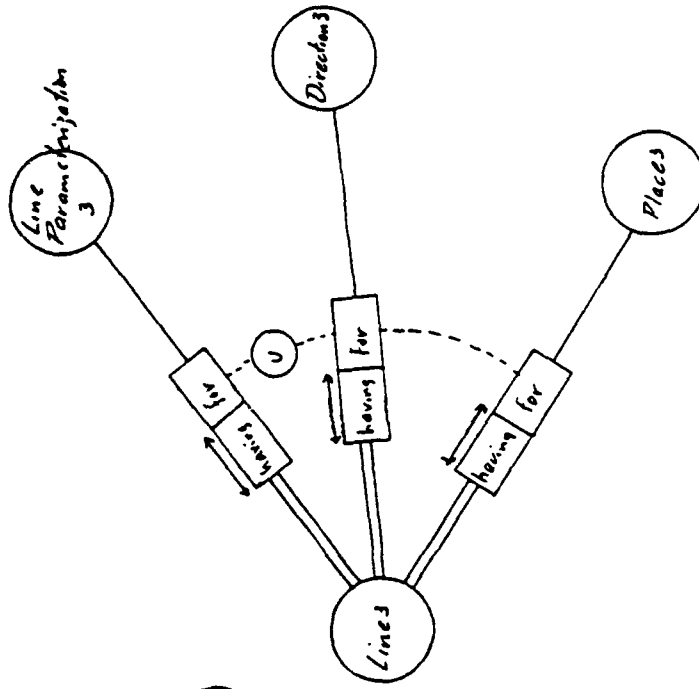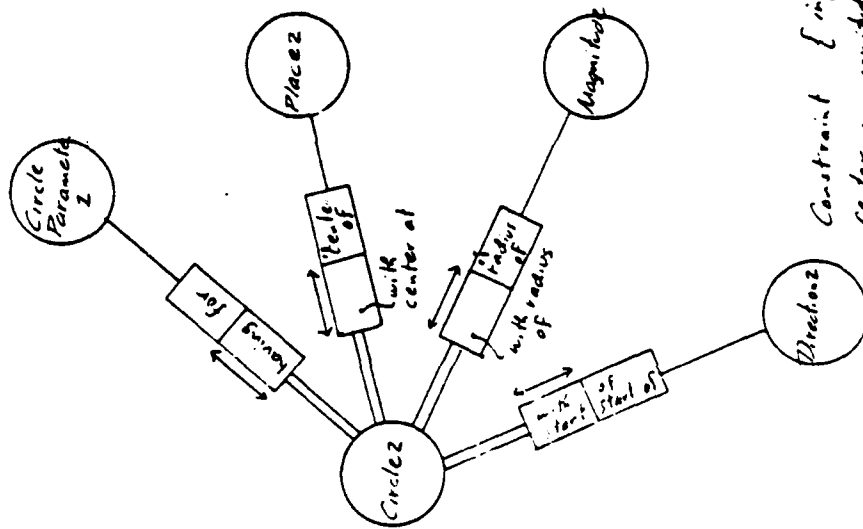
Constraint [informal]
center + magnitude. direction 2
= parameterisation function (0)

Note: Even though 2 circles may share the same point, they are not the same. Assume no joint uniqueness between radius and center.

CURVE SEGMENT

COMPOSITE CURVE

Directions

with i-row vector | For i-row vector in

with j-row vector | For j-row vector in

with k-row vector | For k-row vector in

Transformation matrix

Constraint:

i-row-vector • j-row-vector = 0
j-row-vector • k-row-vector = 0
k-row-vector • i-row-vector = 0

with transform | is transformed to

Places

TRANSFORMATION
MATRIX 2
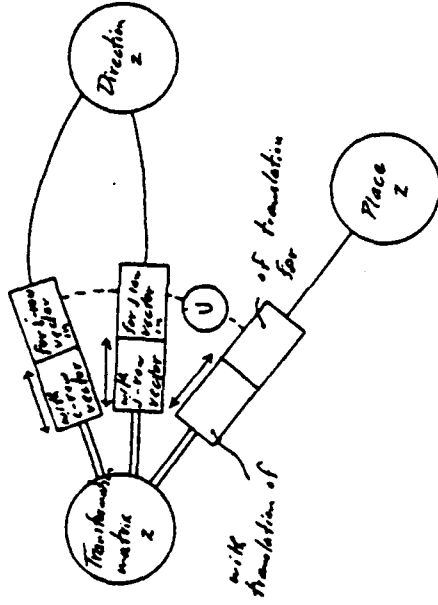
constraint : $i \cdot row\ vector \bullet j \cdot row\ vector = 0$

ELLIPSE 2
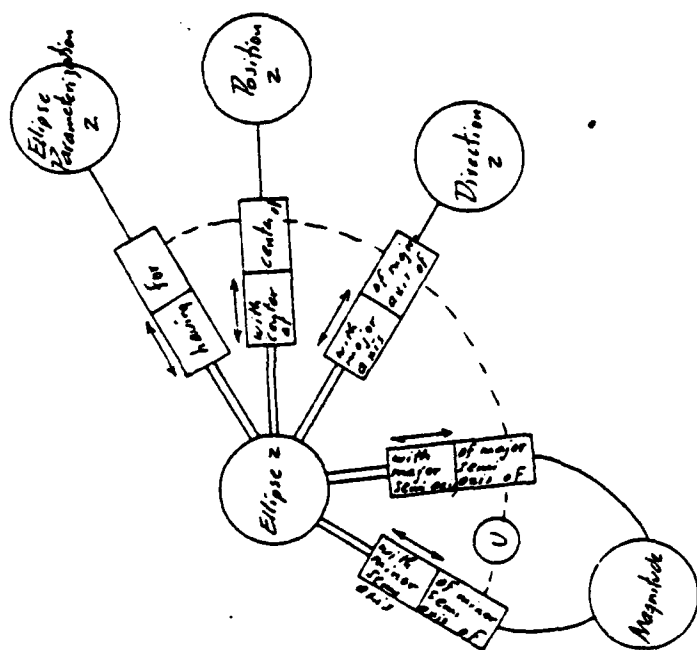
Note : the major axis direction
defines the start of parameterization.

ELLIPSE 3

Ellipse Parameterization 3

Place 3

Direction 3

Ellipse

Magnitude
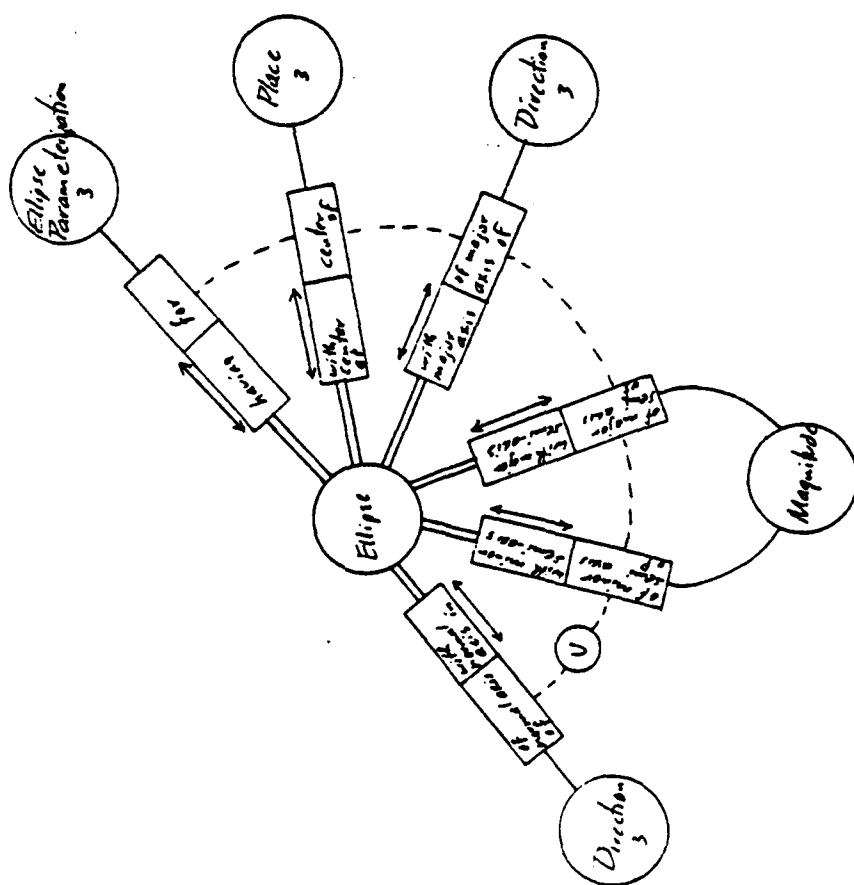
Direction 3

Constraint   Direction-3 of normal axis . Direction-3 of maj axis = 0
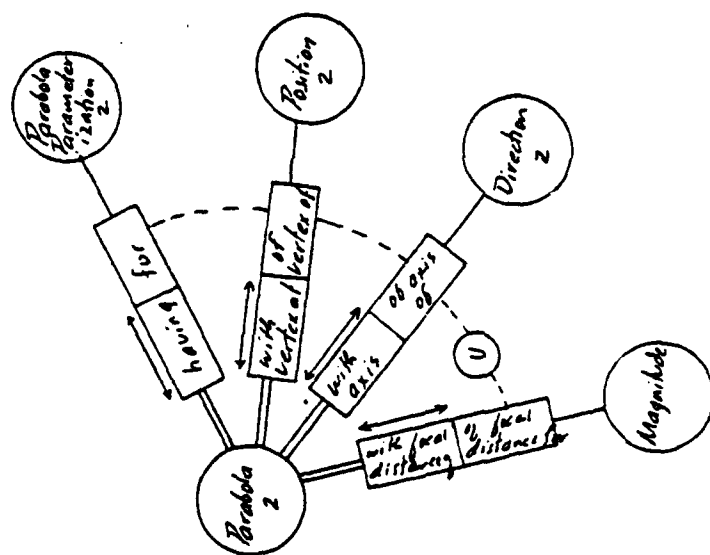
PARABOLA 2

PARABOLA 3
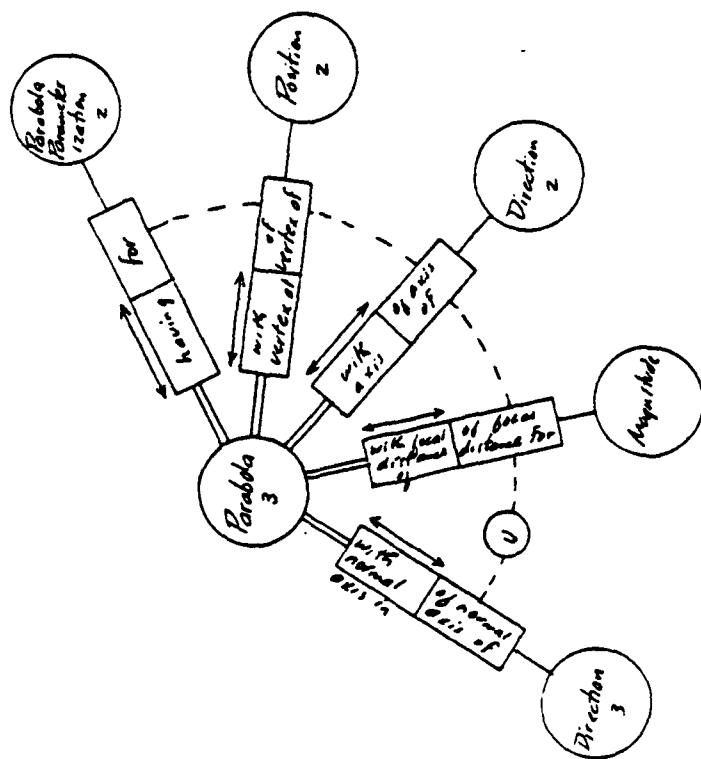
Constraint : Direction 3 of normal axis • Direction 3 of axis = 0

HYPERBOLA 2

Sheet 15.

Note. The selected branch of the hyperbola is pointed to by Direction 2 of major axis
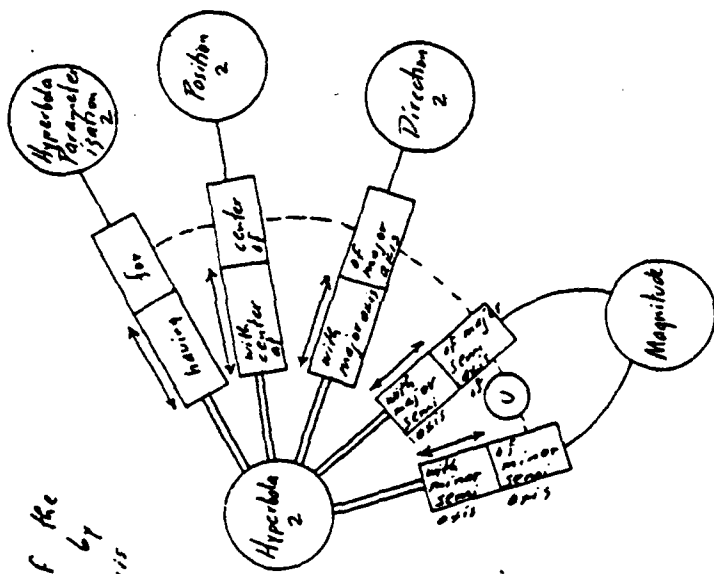
HYPERBOLA 3
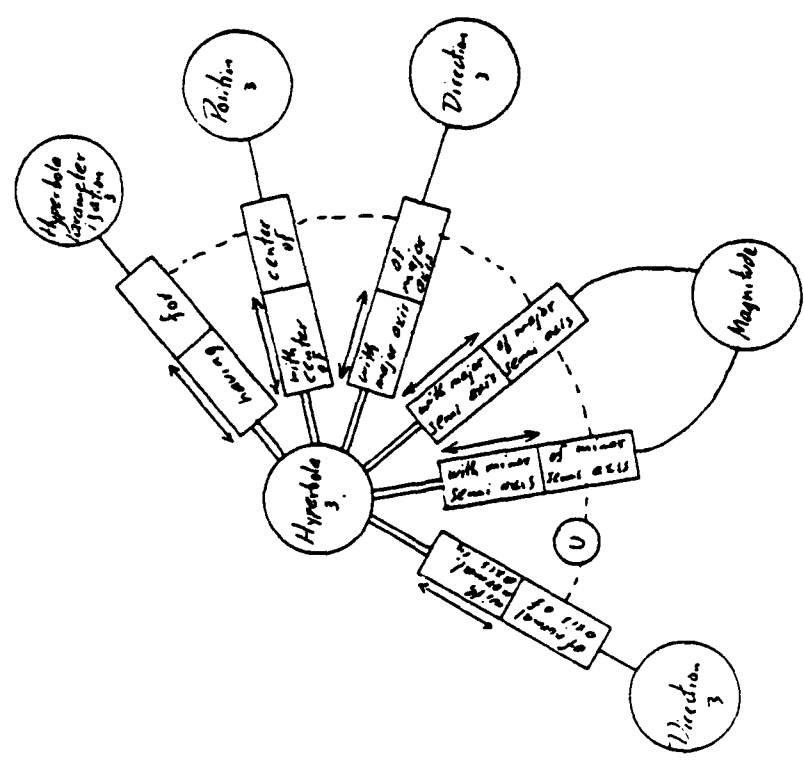
Constraint . Direction 3 of normal axis of . Direction 3 of major axis =0.

SPLINE

Sheet 17
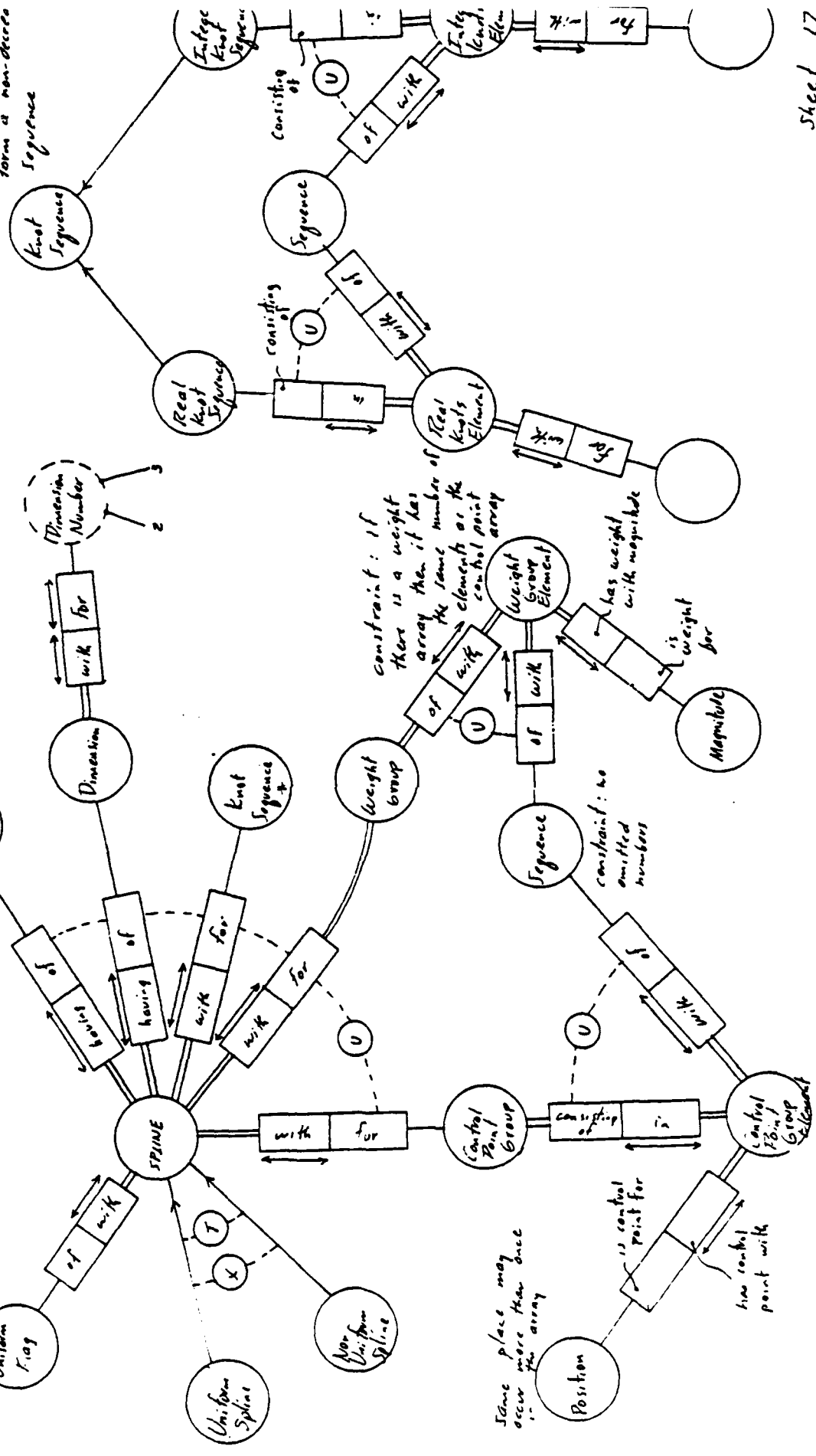
Appendix C1.3

Task 1

Wireframe Geometry Model (DSL)

(RESOURCE)

```
TYPE
   t_continuity = ENUMERATION OF (c0, c1, c2);
      -- used to indicate continuity state at curve junctions.


   t_closure = ENUMERATION OF (open, closed, intersecting);
      -- used to indicate the closure state of a curve.


   t_cart_coord = REAL;
      -- used for position, direction attributes.


   t_magnitude = REAL > 0.0;
      -- used whereever a positive (non-zero) real is required.
      -- the syntax for qualifying values is still up in the air.
      -- I am looking at Chai-Hui Shih's work which covers this
      -- kind of thing quite completely.


   t_bit = LOGICAL;


   t_projection_type_code = ENUMERATION OF(perspective, parallel);


   t_clip_indicator = ENUMERATION OF(clip, noclip);


   t_parallelogram = STRUCTURE;
                                 width : REAL;
                                 length : REAL;
                              END;
```

```
-- START OF WIRE FRAME GEOMETRY ENTITIES
-- VERSION 0.4
-- AUTHOR CLAPP

CLASS wireframe OF
   (wireframe_geometry, wireframe_auxiliary);


CLASS wireframe_geometry OF
   (point, wireframe_segment);


CLASS point OF
   (point2, point3);


CLASS wireframe_segment OF
   (single_segment, composite_segment);


CLASS single_segment OF
   (curve_segment2, curve_segment3);


CLASS composite_segment OF
   (composite_curve2, composite_curve3);


CLASS wireframe_auxiliary OF
   (auxiliary_curve, position, vector, transformation_matrix);


CLASS auxiliary_curve OF
   (curve2, curve3, spline);


CLASS curve2 OF
   (ellipse2, parabola2, hyperbola2, line2, circle2);


CLASS curve3 OF
   (ellipse3, parabola3, hyperbola3, line3, circle3);


CLASS position OF
   (position2, position3);


CLASS vector OF
   (non_unit_vector, direction);
```

```
CLASS non_unit_vector OF
   (vector2, vector3);


CLASS direction OF
   (direction2, direction3);


CLASS transformation_matrix OF
   (trans_matrix2, trans_matrix3);


ENTITY place2;
NOROLE
   name : OPTIONAL string;
ROLE
   x : t_cart_coord;
   y : t_cart_coord;
END;


ENTITY place3;
NOROLE
   name : OPTIONAL STRING;
ROLE
   x : t_cart_coord;
   y : t_cart_coord;
   z : t_cart_coord;
END;


ENTITY point2;
NOROLE
   name : OPTIONAL STRING;
ROLE
   location : REFER( place2 );
END;


ENTITY point3;
NOROLE
   name : OPTIONAL STRING;
ROLE
   location : REFER( place3 );
END;
```

```
ENTITY vector2;
NOROLE
  name : OPTIONAL STRING;
ROLE
  i : t_cart_coord;
  j : t_cart_coord;
  l : t_magnitude;
END;


ENTITY vector3;
NOROLE
  name : OPTIONAL STRING;
ROLE
  i : t_cart_coord;
  j : t_cart_coord;
  k : t_cart_coord;
  l : t_magnitude;
END;


ENTITY direction2;
NOROLE
  name : OPTIONAL STRING;
ROLE
  i : t_cart_coord;
  j : t_cart_coord;
END;


ENTITY direction3;
NOROLE
  name : OPTIONAL STRING;
ROLE
  i : t_cart_coord;
  j : t_cart_coord;
  k : t_cart_coord;
END;


ENTITY line2;
NOROLE
  name : OPTIONAL STRING;
ROLE
  place : REFER( place2 );
  direction : REFER( direction2 );
END;
```

```
ENTITY line3;
NOROLE
  name : OPTIONAL STRING;
ROLE
  place : REFER( place3 );
  direction : REFER( direction3 );
END;


ENTITY circle2;
NOROLE
  name : OPTIONAL STRING;
ROLE
  center : REFER( place2 );
  radius : t_magnitude;
  start : REFER( direction3 );
END;


ENTITY circle3;
NOROLE
  name : OPTIONAL STRING;
ROLE
  axis : REFER( direction3 );
  center : REFER( place3 );
  radius : t_magnitude;
  start : REFER( direction3 );
END;


ENTITY ellipse2;
NOROLE
  name : OPTIONAL STRING;
ROLE
  center : REFER( place2 );
  majoraxis : REFER( direction2 );
  semimajor : t_magnitude;
  semiminor : t_magnitude;
END;


ENTITY ellipse3;
NOROLE
  name : OPTIONAL STRING;
ROLE
  center : REFER( place3 );
  majoraxis : REFER( direction3 );
  semimajor : t_magnitude;
  semiminor : t_magnitude;
  normalaxis : REFER( direction3 );
END;
```

```
ENTITY parabola2;
NOROLE
  name : OPTIONAL STRING;
 OLE
  vertex : REFER( place2 );
  axis : REFER( direction2 );
  focaldist : t_magnitude;
END;


ENTITY parabola3;
NOROLE
  name : OPTIONAL STRING;
ROLE
  vertex : REFER( place3 );
  axis : REFER( direction3 );
  focaldist : t_magnitude;
  normalaxis : REFER( direction3 );
END;


ENTITY hyperbola2;
NOROLE
  name : OPTIONAL STRING;
ROLE
  center : REFER( place2 );
  majoraxis : REFER( direction2 );
  semimajor : t_magnitude;
  semiminor : t_magnitude;
END;


ENTITY hyperbola3;
NOROLE
  name : OPTIONAL STRING;
ROLE
  center : REFER( place3 );
  majoraxis : REFER( direction3 );
  semimajor : t_magnitude;
  semiminor : t_magnitude;
  normalaxis : REFER( direction3 );
END;
```

```
ENTITY spline;
NOROLE
  name : OPTIONAL STRING;
ROLE
  order : INTEGER WHERE RANGE 2 TO MAXINT;
  dimension : INTEGER WHERE RANGE 2 TO 3;
  uniform : LOGICAL;
  controlpoint : ARRAY(0:*) OF REFER( position );
  weight : OPTIONAL ARRAY(0:UBOUND(controlpoint)) OF REAL;
  knotsequence : ARRAY(-(order-1):(UBOUND(controlpoint)+1)) OF
                       NUMBER;
END;
-- NOTE: UBOUND is a built-in function which returns an integer
-- number which is the upper bound of an array.

ENTITY curve_segment2;
NOROLE
  name : OPTIONAL STRING;
ROLE
  base : REFER( curve2d );
  path : LOGICAL;
  startpoint : REFER( place2 );
  endpoint : REFER( place2 );
  pstart : REAL;
  pend : REAL;
END;


ENTITY curve_segment3;
NOROLE
  name : OPTIONAL STRING;
ROLE
  base : REFER( curve3d );
  path : LOGICAL;
  startpoint : REFER( place3 );
  endpoint : REFER( place3 );
  pstart : REAL;
  pend : REAL;
END;


ENTITY composite_curve2;
NOROLE
  name : OPTIONAL STRING;
ROLE
  closure : t_closure;
  degree : t_continuity;
  constituent : LIST(2 TO *) OF
    STRUCTURE;
      element : REFER( curve_segment2 );
      sense : LOGICAL;
      degree : t_continuity;
    END;
END;
```

```
ENTITY composite_curve3;
NOROLE
  name : OPTIONAL STRING;
ROLE
  closure : t_closure;
  degree : t_continuity;
  constituent : LIST(2 TO *) OF
    STRUCTURE;
      element : REFER( curve_segment3 );
      sense : LOGICAL;
      degree : t_continuity;
    END;
END;


ENTITY trans_matrix2;
NOROLE
  name : OPTIONAL STRING;
ROLE
  irow : REFER( direction2 );
  jrow : REFER( direction2 );
  translation : REFER( place2 );
END;


ENTITY trans_matrix3;
NOROLE
  name : OPTIONAL STRING;
ROLE
  irow : REFER( direction3 );
  jrow : REFER( direction3 );
  krow : REFER( direction3 );
  translation : REFER( place3 );
END;
```

Appendix C2

Task 1

(RESOURCE)

<u>Presentation Resource Model</u>

C2.1   Initial Model and Documentation

C2.2   NIAM Model

C2.3   DSL Model

Appendix C2.1

Task 1

<u>Presentation Initial Model and Documentation</u>

(RESOURCE)

Richard C. Winfrey
Consulting Engineer
CIM Technology
Digital Equipment Corporation
SHR1-3/E10
Shrewsbury, Massachusetts 01545-4112
(617) 841-2192

November 4, 1985

Introduction

The following is a list of comments by reviewers of the DRAFT proposal dated June 14, 1985. I would like to thank them for their time in reviewing the document and for their perceptive comments. Following many of the comments, I have tried to offer an answer, or add further explanation, and these are marked as "RCW response:". These are strictly my personal comments and are not necessarily those of the PDES/IGES committee.

The purpose of the DRAFT proposal was to specify the presentation needs of PDES. Once those needs had been defined, with the help of the PHIGS, GKS-3D, and CG-VDI proposals, the result was very nearly that of PHIGS. Therefore, rather than spend additional time (redundant) refining the DRAFT proposal, the following recommendations are made:

o Identify the few additional needs of PDES that are not included in PHIGS, and offer them to the PHIGS committee for inclusion in their standard.

o With the help of the PHIGS committee, define a PDES level of implementation. The entire scope of the PHIGS functionality is not needed for PDES.

o Create a new document having a title something like: "PDES Implementor's Guide to the use of PHIGS". This would answer many of the questions that have been raised by the PDES/IGES committee members as well as provide a convenient place to keep examples and recommended practices that relate to graphics. Typical questions might be; "How do I set up the viewing pipeline?", "How do I set up the pipeline to create a drawing or views?", or "How do I use the pipeline to provide for scaling?".

Issues

1. Alan Peltzman, Inter CAD Corp., (301) 224-2926

    a. On page 4, the diagram incorrectly shows the origin of the UVW coordinate system to be different from the view reference point. This contradicts the rest of the document, which states that the origin of the UVW is the View Reference Point.

       RCW response: You're right, thanks.

    b. On page 5, section 2.2.2, the N-axis needs to be explicitly defined.

       RCW response: See "Implementor's Guide".

    c. The view matrix on page 5, section 2.2.4, should be eliminated. Redundant information.

       RCW response: This was included for computational efficiency. It is certainly redundant and could be eliminated.

    d. On page 8, section 2.3.3, UMIN, UMAX, VMIN, and VMAX need to be explicitly defined.

       RCW response: See "Implementor's Guide".

    e. As a side issue, please ask PHIGS how they propose to deal with light source information. This is very important in shading applications.

       RCW response: This will be part of the comments to PHIGS.

2. Don Hemmelgarn, International TechneGroup Inc., (513) 576-3900

    a. Requirement for Drafting (Drawing) Coordinates.

       There is a required presentation concept which is not addressed by the document "DRAFT of Presentation Entities for PDES." This is the concept of an independent coordinate space for the location of views and the definition of drawing entities. This "drawing space" coordinate system will need only to be a 2-D coordinate system. One requirement for this capability would be the specification of the extents of the drawing or drawing size.

       One possible way to do this in the context of the given viewing pipeline would be to locate views and specify drawing entities in NPC coordinates. This would then

allow the NPC space to map exactly to the drawing. This would require, however, that NPC extents be made more flexible by allowing 0.0 to 1.0 in one direction and 0.0 to some value in the other direction to allow rectangular drawings.

RCW response: See "Implementor's Guide".

b. Scaling

The document does not include any details on view scaling. It would be helpful to see examples of how view scaling is accomplished in the pipeline, in addition to how a traditional view matrix and scale factor would be mapped into the proposed viewing pipeline.

RCW response: See "Implementor's Guide".

c. Device Coordinates

My feeling is that the Display Transformation need not be specified or included as part of a product data exchange file. For a post-processor, it is better to work with NPC coordinates and map these to the particular device configuration that he uses.

RCW response: My feeling is that the specification of the complete viewing pipeline from model coordinates to the display surface must be provided for. However, remember that since the Presentation is not actually part of the Product Definition, it can be treated optionally. Thus, if a vendor wishes to work with NPC coordinates and ignore the Device Transformation, that is his prerogative. The important point is that the specification is complete if the user wants to use it.

d. Character Set

It should be mentioned that the ASCII character set is to be used in defining text strings. For clarity, we should spell out the difference between character set and text font. These concepts have been muddled in IGES. Text font is merely a way of presenting the characters in the defined character set. Also, there will need to be a set of special symbols (to be defined by drafting application and others) added to the standard character set.

RCW response: See "Implementor's Guide".

3. **Martin Marietta Aerospace, Orlando, FL**

    a. **Optional Presentation Section.**

        Will the entire Presentation section be made optional in
        a file? This would be desirable from the standpoint of
        passing product definition data from design to
        manufacturing, where display characteristics and
        "drawing" information such as formats are not needed.
        If this is the case, presentation entities should not
        contain information needed in geometry/manufacturing
        instructions such as text.

        RCW response: As presently envisioned, Presentation
        entities are not mandatory; however, they will remain
        quite important for those who use PDES in the same
        fashion as they are now using IGES. With regard to
        text, it must exist both in and out of the presentation
        entities. Text such as dimensions and notes that are
        part of a drawing are, by definition, part of the
        presentation set, while text such as process plans and
        technical documents are not. It would probably be a
        good idea to allow the text string (T_STR in the TEXT
        entity) to be optionally a pointer so notes and comments
        that are part of the product definition could be
        referenced and displayed on the screen or a drawing.

    b. **Colors and surface attributes.**

        The color table and color attribute appear to be
        redundant and also the color table will not allow
        precise color specification. See Ted Berenyi's work on
        RGB vs HSL, etc. Again, colors should be assignable to
        bodies in solids; therefore, they should be available
        outside the presentation part. The same argument
        applies to filling. The fill options must also be
        expanded eventually to include surface texture display
        attributes, as well as glossiness, translucency/
        transparency and light source information.

        RCW response: It is generally agreed that the HLS color
        model is more "convenient" for people and the RGB model
        is more convenient for computer hardware. Since one can
        easily convert back and forth between the two, it was
        decided to specify only one in the interest of
        simplicity. Since the PDES file is presumably to
        transfer data between computer hardware rather than
        people, the RGB model was selected. The COLOR_ATTR
        entity (para. 4.9) provides a means for specifying a
        single color (to about 6 or 7 decimal places for each of
        the red, green, and blue components). The COLOR_TABLE
        entity acknowledges that display hardware can only
        display a limited number of different colors at any one
        time, and that these displayable colors are typically
        defined in a hardware color look-up table. The number

of different colors depends on the number of pixel planes in the hardware and on how these planes are used. Hence, each index (IND_1...IND_N) is a pointer to a specific color - each of which is defined with more precision than is available in the hardware.

With regard to assigning colors, fill patterns, etc. to geometrical entities, we must remember that the object of PDES is to exchange product definition data. By contrast, Presentation is intended to aid in viewing a model, but in general has nothing to do with its definition. Hence, it is important to separate the two and not allow them to become intermixed as they are in IGES. On the other hand, attributes such as surface texture, glossiness, and translucency/transparency could easily be either, or both, Product Definition and Presentation. These latter attributes, along with light source (which, again, is Presentation), were discussed by the Logical Layer Group, and it was decided to add them at a later time. It is now recommended that they be included in a recommendation to PHIGS.

CONTENTS

DRAFT of Presentation Entities for PDES

Richard C. Winfrey
Consulting Engineer
CIM Technology
Digital Equipment Corporation
SHR1-3/E18
Shrewsbury, Massachusetts   01545-4112
(617) 841-2192

November 4, 1985

DRAFT of Presentation Entities for PDES

Richard C. Winfrey
Consulting Engineer
CIM Technology
Digital Equipment Corporation
SHR1-3/E10
Shrewsbury, Massachusetts 01545-4112
(617) 841-2192

November 4, 1985

1 INTRODUCTION

The following is a proposal for a set of entities to be used in the presentation of graphical data within PDES. The objective is to allow the "state of the viewing screen" in one CAD system to be transferred to another CAD system so that viewers on both systems can not only look at the same data, but can look at in the same WAY.

What we are going to implement, in essence, is a small part of a graphics specification, and it is only reasonable to investigate currently available standards. Unfortunately, there ARE no 3D graphics standards (the current GKS standard is only 2D). Thus, the following is a result of a careful review of the currently available specifications; namely PHIGS, GKS-3D, CG-VDI, and CORE. I have also included two of the major references in the literature (see References).

To be sure, PHIGS, GKS-3D, and CORE have substantial differences in their entirety. However, we are only concerned here with a very narrow and restricted portion of these specifications, primarily the viewing pipeline. In this regard, one finds a fairly consistent and steady philosophy, starting in the early 1970's and culminating with PHIGS, as to how a viewing pipeline should be defined. In addition, there is a conscious effort to minimize the differences between the PHIGS and GKS-3D proposals.

Primary differences, as they effect PDES, are that GKS-3D does not provide for a Local-to-World transformation that we currently make use of in IGES, and there is no provision for a hierarchical data structure. Otherwise, if in GKS-3D, one sets the first normalizing transformation to an identity matrix and ignores the first clipping operation, then the viewing pipeline becomes essentially identical with that of PHIGS.

Therefore, I have tended to use the PHIGS document in what is to follow because I feel it is clear that PHIGS most closely

matches the needs of PDES. However, because of the above described similarities between PHIGS and GKS-3D, I feel that implementing these Presentation entities in a system using a GKS-3D or CORE based graphics system will involve little or no additional effort.

One exception to the use of PHIGS is with the subject of text. It was felt that the CG-VDI proposal contained material that was more suitable to the needs of PDES. Therefore, the CG-VDI proposal supplied much of the information that is given below in the section on text.

2 TRANSFORMATIONS

Graphical information is transferred from a data structure to a display screen by means of a series of coordinate transformations, and it is the purpose of this proposal to define this step-by-step procedure. More specifically, it is the intent to identify the parameters that make up these transformations. If necessary, implementation guidelines can be included at a later date.

The viewing pipeline as proposed by PHIGS is shown in Figure 1. The following sections define the parameters necessary to step from one coordinate system to the next. Bear in mind that these steps are defined seperately for conceptual reasons, and that during implementation they are often combined for computational efficiency. Four transformations are defined as well as five coordinate systems, all right handed.

```
+----------------------+
| 3-D Output Elements  |     <--- Modeling Coordinates (MC)
+----------------------+
+----------------------+
| Composite Modeling   |
|    Transformation    |
+----------------------+
                             <--- World Coordinates (WC)
+----------------------+
|      Viewing         |
|   Transformation     |
+----------------------+
                             <--- Viewing Coordinates (VC)
+----------------------+
|     Clipping         |
|  Operation and       |
|   View Mapping       |
+----------------------+
                             <--- Normalised Projection
                                  Coordinates (NPC)
+----------------------+
|      Display         |
|   Transformation     |
+----------------------+
                             <--- Device Coordinates (DC)
+----------------------+
|      Display         |
+----------------------+
```
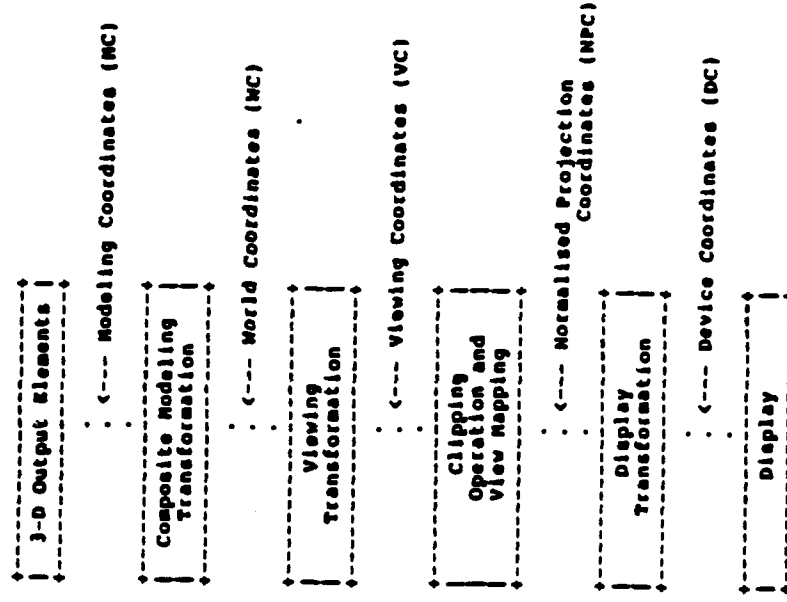
Figure 1. Transformations and Coordinate Systems

2.1 Composite Modeling Transformation

This transformation is the same transformation that is presently specified in IGES, and transforms from Local coordinates to World coordinates. The assumption is that this will be defined elsewhere in PDES and need not be duplicated here. The application supplies a 4x3 matrix of type float, which is implicitly expanded to a 4x4 matrix. The default (implicit) is the 4x4 unit matrix.

2.2 Viewing Transformation

The viewing utility functions provide a means to calculate a Viewing Transformation Matrix, the 4x4 matrix which maps World Coordinates to the UVN Viewing Coordinate System. The Viewing Transformation Matrix is defined by the following 3 entities (see Figure 2).
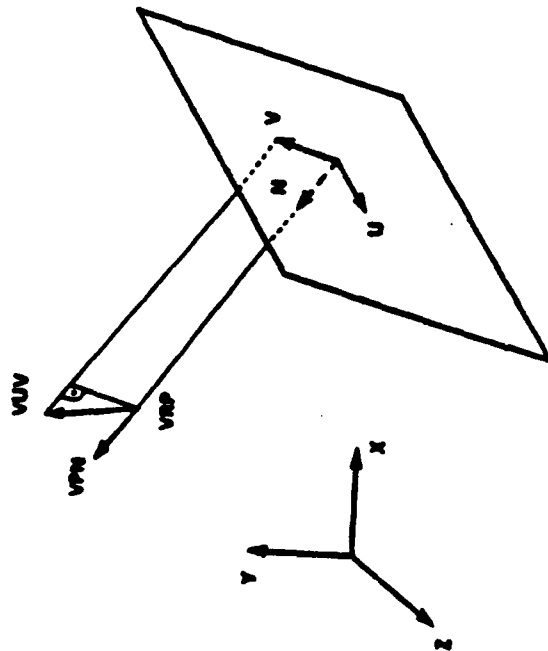
Figure 2. Orientation of View Coordinate System

2.2.1 View Reference Point -

The VIEW REFERENCE POINT is a World Coordinate location which serves as a reference point for the construction of the viewing system and is the origin of the viewing coordinate system. The View Reference Point is typically a point on or near the object to be viewed.

```
VIEW_REF_POINT - entity_type

    Name    : Optional string;
    VRP     : Pointer (Point3);
    End;
```

Default value is (0,0,0), the origin of the World Coordinate System.

2.2.2 View Plane Normal -

The VIEW PLANE NORMAL is a world coordinate vector which is relative to the View Reference Point. This vector describes an infinite family of parallel planes, one of which serves as the View Plane upon which the model is projected.

```
VIEW_PLANE_NORMAL - entity_type

    Name    : Optional string;
    VPN     : Pointer (Vector);
    End;
```

Default value is (0,0,1).

2.2.3 View Up Vector -

The VIEW UP VECTOR is a world coordinate vector which is relative to the View Reference Point. This vector is projected onto the View Plane with a projection parallel to the View Plane Normal. The projection of the View Up Vector onto the View Plane determines the V axis of the UVN system. The U axis is 90 degrees clockwise from the V axis when viewed in the negative N direction (down the negative View Plane Normal direction). The UVN system is a right handed coordinate system.

```
VIEW_UP_VECTOR - Entity_type

    Name    : Optional string;
    VUP     : Pointer (Vector));
    End;
```

Default value is (0,1,0).

2.2.4 View Matrix -

The VIEW MATRIX is computed from the VIEW_REF_POINT, the VIEW PLANE NORMAL, and the VIEW UP VECTOR (see Ref 4, UVN coordinates, but note that the VC are right-handed in this specification and left-handed in reference 4). It is required that the above 3 entities be specified; however the View Matrix is optional - it is often useful to have it explicitly given for computational efficiency. The pointers are optional, and

Figure 4. Perspective Projection

2.3.1 Projection Type -

This specification supports two types of projections: parallel projection and perspective projection.

PROJECTION_TYPE = entity_type

 Name   : Optional string;
 PROJ_TYPE : String;
 End;

PROJ_TYPE must have the value of 'PARALLEL' or 'PERSPECTIVE'. The default value is PARALLEL.

2.3.2 View Plane Distance -

The View Plane Distance locates the view plane at a signed distance from the origin of the viewing coordinate system along the N axis. The view plane is, by definition, parallel to the u-v plane.

---

omission indicates that the current value (default value if not previously specified) is to be used.

VIEW_MATRIX = Entity_type

 Name : Optional string;
 VRP : Optional Pointer (VIEW_REF_POINT);
 VPN : Optional Pointer (VIEW_PLANE_NORMAL);
 VUP : Optional Pointer (VIEW_UP_VECTOR);
 End;

Default value is 4x4 identity matrix.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2.3 View Mapping Functions.

The View Mapping transformation performs a window to viewport transformation which maps the Viewing Coordinate system to the Normalized Projection Coordinate space (see Figures 3 and 4).

Figure 3. Parallel Projection

2.3.6 Projection Reference Point -

The projection reference point, (a point in UVN space) orients the projectors in the viewing coordinate system. In a parallel view, the projection reference point and the center of the window form a direction vector to which all projectors are parallel. An oblique view results when the U and V components of the projection reference point have values other than the components of the window center. In a perspective view, the projection reference point is the point where all projectors converge. A non-centered view (such as might be used for stereoscopic presentation) is achieved by specifying the window such that the center of the window is not at (0,0) on the U-V plane.

```
PROJ_REF_POINT  - entity_type

    Name : Optional String;
    PRP  : Pointer (Point3];
    End;
```

2.3.7 NPC Viewport -

The 3D viewport defines a rectangular parallelepiped in NPC space into which the 3D window volume is mapped. The window volume is mapped into the viewport in NPC such that the part of the front plane defined by the window volume coincides with the viewport zmax-plane and the part of the back plane defined by the window volume coincides with the viewport zmin-plane.

```
NPC_VIEWPORT  - entity_type

    Name : Optional String;
    XMIN : float;
    XMAX : float;
    YMIN : float;
    YMAX : float;
    ZMIN : float;
    ZMAX : float;
    End;
```

2.3.8 Clipping Indicators -

Clipping may be controlled separately for the UV window, the front clipping plane, and the back clipping plane.

```
VIEW_PLANE_DISTANCE     - entity_type

    Name : Optional String;
    VPD  : Float;
    End;
```

Default distance is 0.

2.3.3 View Plane Window -

The 3D window volume is defined by a window on the view plane specified by (UMIN, UMAX, VMIN, VMAX) and front and back clipping planes specified by (NMAX, NMIN). The window volume is formed by the window on the view plane, the front and back clipping planes, and the projectors used in the selected projection type. The parallel projectors of a parallel view result in a window volume which is a rectangular parallelepiped. The converging projectors of a perspective view result in a window volume which is a truncated frustum or double pyramid depending on the placement of the front and back planes.

```
UV_WINDOW     - entity_type

    Name : Optional String;
    UMIN : Float;
    UMAX : Float;
    VMIN : Float;
    VMAX : Float;
    End;
```

2.3.4 Front Clipping Plane -
```
FRONT_CLIPPING_PLANE     - entity_type

    Name : Optional String;
    NMAX : Float;
    End;
```

2.3.5 Back Clipping Plane -
```
BACK_CLIPPING_PLANE     - entity_type

    Name : Optional String;
    NMIN : Float;
    End;
```

WS_WINDOW   = entity_type
    Name   : Optional String;
    XMIN   : Float;
    XMAX   : Float;
    YMIN   : Float;
    YMAX   : Float;
    ZMIN   : Float;
    ZMAX   : Float;
    End;

2.4.2 Device Viewport -
    DC_VIEWPORT   = entity_type
    Name   : Optional String;
    XMIN   : Float;
    XMAX   : Float;
    YMIN   : Float;
    YMAX   : Float;
    ZMIN   : Float;
    ZMAX   : Float;
    End;

3 PRESENTATION FUNCTIONS

1.1 Picture

The end result of presentation is a picture on the screen. This picture is created by a hierarchy of Presentation Lists to be defined, and the Picture entity is used here to define the start, or root, of a hierarchical tree. It may also be used to set initial display values.

PICTURE   = entity_type
    Name   : Optional String;
    VU_OPR   : Optional Pointer   (VIEWING OPER);
    WS_TRM   : Optional Pointer   (WORKSTATION_TRANS);
    LIN_ATR   : Optional Pointer   (LINE_ATTR);
    SRF_ATR   : Optional Pointer   (SURFACE ATTR);
    TXT_ATR   : Optional Pointer   (TEXT ATTR);
    PRES   : Pointer (PRESENT_LIST);   Root of hierarchy.
    End;

CLIPPING_INDICATORS   = entity_type
    Name   : Optional String;
    WINDOW_CLIP   : Optional String;
    FRONT_CLIP   : Optional String;
    BACK_CLIP   : Optional String;
    End;

The only permissible values are 'CLIP and 'NOCLIP'.
The default for all is 'CLIP'.

2.4 Workstation Functions

The NPC space can be regarded as a device dependent abstract viewing space. The application can select some part of the NPC space in the range [0,1]x[0,1]x[0,1] to be displayed somewhere on the display surface. This transformation is an isotropic mapping from NPC space onto the device coordinates (scale factor in the X and Y directions is the same).

The workstation transformation is specified by defining the limits of a volume in NPC space within the range [0,1]x[0,1]x[0,1] (the workstation window), which is to be mapped onto a specified volume of DC space (the workstation viewport). Workstation window and workstation viewport limits specify rectangular boxes parallel to the coordinate axes in NPC and DC. Output is clipped at the workstation window boundaries, and cannot be disabled for X and Y. Z-clipping is specified by the same clipping indicators defined in the previous section.

If the workstation window and workstation viewport have different aspect ratios, the transformation maps the workstation window into the largest rectangular box that can fit within the workstation viewport such that:

1. Ratio is preserved.
2. The lower, left front corner of the workstation window is mapped to the lower, left, front corner of the workstation viewport.

2.4.1 Workstation Window -

The default is to map the entire NPC space [0,1]x[0,1]x[0,1] onto the full display surface consistent with the rules for different aspect ratios as described above.

3.2 Presentation List

The Presentation List (display list structure) contains a list of entities to be displayed. The list may contain other Presentation List entities, but it may not be used recursively - i.e., it may not refer to itself either directly or indirectly by means of one or more other lists.

The Presentation List is displayed by sequentially displaying each entity in the list, beginning with the first. When another Presentation List entity is encountered, the following actions occur:

1. Display of the current list is suspended.

2. The state of the attribute values is saved.

3. The new list is displayed - including all the Presentation Lists that it may contain.

4. The attribute values are restored to the state saved (from step 2.) before the new list was displayed.

5. Display of the parent list (from step 1.) is resumed.

This means:

1. At display time, a Presentation List may inherit the attributes of those lists to which it is subordinate in the hierarchy.

2. At display time, a Presentation list may affect only those lists subordinate to it in the hierarchy.

```
PRESENT_LIST    - entity_type

    Name    *: Optional String;
    VU_OPR  : Optional Pointer  (VIEWING OPER);
    WS_TRN  : Optional Pointer  (WORKSTATION_TRANS);
    LIN_ATR : Optional Pointer  (LINE ATTR);
    SRF_ATR : Optional Pointer  (SURFACE ATTR);
    TXT_ATR : Optional Pointer  (TEXT ATTR);
    LENG    : Integer;
    E_1     : Pointer to first entity;

    E_LENG  : Pointer to last entity;
    End;
```

LENG - Number of entities in the list. If .le. 0, this Presentation List entity is ignored, and control is returned to

---

the parent list, if any.

3.3 Viewing Operations (view Table)

```
VIEWING_OPER         - entity_type

    Name    : Optional string;
    VIEW    : Optional Pointer  (VIEW MATRIX);
    WIN     : Optional Pointer  (UV WINDOW);
    VP      : Optional Pointer  (NPC VIEWPORT);
    PRJ_TYP : Optional Pointer  (PROJECTION TYPE);
    PRP     : Optional Pointer  (PROJ REF POINT);
    VPD     : Optional Pointer  (VIEW PLANE DISTANCE);
    NMIN    : Optional Pointer  (BACK CLIPPING PLANE);
    NMAX    : Optional Pointer  (FRONT CLIPPING PLANE);
    CLIP    : Optional Pointer  (CLIPPING_INDICATORS);
    End;
```

3.4 Workstation Transformation

```
WORKSTATION_TRANS    - entity_type

    Name    : Optional string;
    WIN     : Optional Pointer  (WS_WINDOW);
    VP      : Optional Pointer  (DC_VIEWPORT);
    End;
```

4 TEXT ATTRIBUTES

4.1 Font Coordinate System

The font coordinate system is illustrated in Figure 5. The character body normally encloses all of the drawn parts (kerning excepted) of all characters in the font (that is, no descender extends lower than 'bottom', and no accent mark or oversized symbol extends higher than 'top'). The left and right edges of the character body may be defined on a per-character basis to accommodate variable widths, and proportional spacing. The body exceeds the actual character symbol width and height as necessary to provide adequate white space between characters both vertically and horizontally, such that the text is readable and adequately separated when adjacent character bodies are flush (that is, when CHARACTER SPACING is 0) and multiline text can be created using TEXT ALIGNMENT without causing vertical overlaps. It is expected that font designers will specify some fonts having kerns extending beyond that character body horizontally; however, the RESTRICTED TEXT entity still requires that all visible portions of the character be confined to the text restriction rectangle.
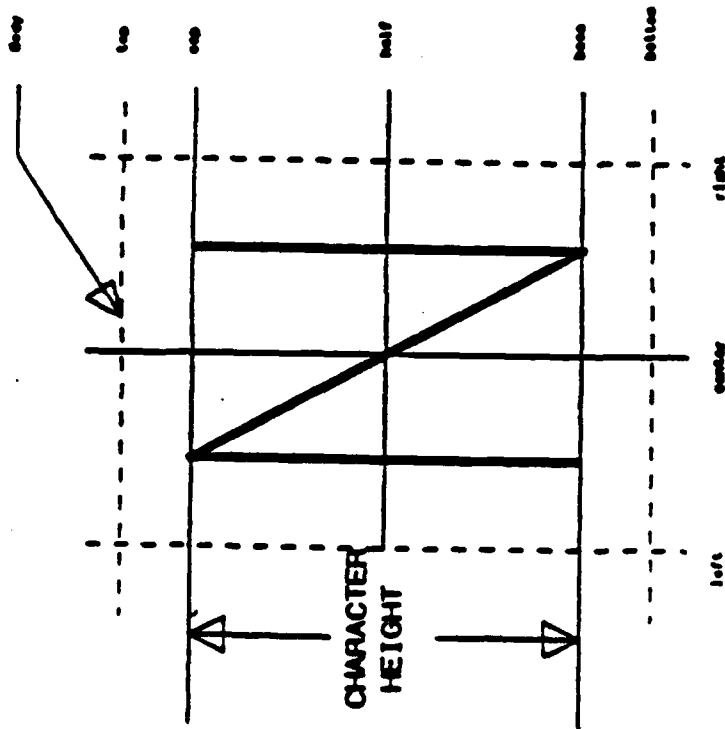
CHARACTER
HEIGHT

Figure 5. Font Description Coordinate System

4.2 Text Functions

The following aspects define the representation of a TEXT primitive: TEXT FONT INDEX, TEXT PRECISION, CHARACTER EXPANSION FACTOR, CHARACTER SPACING, TEXT COLOR INDEX, CHARACTER HEIGHT, CHARACTER ORIENTATION, TEXT PATH, and TEXT ALIGNMENT.

Precise control of the appearance of TEXT on a workstation is provided by the following aspects: CHARACTER HEIGHT, CHARACTER EXPANSION FACTOR, CHARACTER SPACING, CHARACTER ORIENTATION, TEXT PATH, and TEXT ALIGNMENT. However, the use of these values in displaying text is determined by the setting of the TEXT FONT INDEX and TEXT PRECISION aspects. When the text is viewed in perspective, each character may have a different height. When

---

not in stroke precision, the character height to be used is determined by using the leading edge of each character for CHAR precision.

The attributes in the character representation and placement group and TEXT_ATTR_INDEX may be changed within a string. A TEXT entity or RESTRICTED_TEXT entity is flagged to show it is not complete and provides only the first portion of the string. The TEXT entity or RESTRICTED_TEXT entity may be followed by the desired text attribute entities and then by an APPEND_TEXT entity, which provides the next portion of the string. This may be repeated as often as necessary, with the final APPEND_TEXT flagged to indicate that the string is complete. Note that such text generally cannot be displayed until the string is complete because of TEXT ALIGNMENT and the way in which attribute changes affect the definition of the text extent rectangle (see below). Text may be displayed before the string is complete only in the following cases:

PATH	VERTICAL_ALIGNMENT	HORIZONTAL_ALIGNMENT
RIGHT	NORMAL_VERTICAL or BASELINE	NORMAL_HORIZONTAL, LEFT, or CONTINUOUS (0,0)
LEFT	NORMAL_VERTICAL or BASELINE	NORMAL_HORIZONTAL, RIGHT, or CONTINUOUS (1,0)
DOWN	TOP, CAPLINE, NORMAL_VERTICAL, or CONTINUOUS (0,1)	NORMAL_HORIZONTAL or CENTER
UP	BASELINE, BOTTOM, NORMAL_VERTICAL, or CONTINUOUS (0,0)	NORMAL_HORIZONTAL or CENTER

4.2.1 CHARACTER HEIGHT -

The CHARACTER HEIGHT specifies the distance, in modeling coordinates, between CAPLINE and BASELINE of the font (see Figure 6) measured along the CHARACTER_UP_VECTOR. (Note that if the CHARACTER_UP_VECTOR is skewed, the baseline-to-capline perpendicular distance will be less than the CHARACTER_HEIGHT.)

CHAR_HEIGHT   = entity_type

    Name   : Optional string;
    Ht   : Float; non-negative.
    End;

Default value is 0.1

of the CHARACTER_UP_VECTOR.

```
CHAR_EXPAN      = entity_type

    Name      : Optional string;
    CEF       : Float;
    End;
```

Default value is 1.0

4.2.3 CHARACTER SPACING -

CHARACTER SPACING specifies how much additional space is to be inserted between two adjacent character bodies (see Figure 7). If the value of CHARACTER SPACING is zero the character bodies are arranged one after the other along the TEXT PATH with only the intercharacter spacing designated by the font designer. If the value of CHARACTER SPACING is positive, additional space is inserted between character bodies. If the value of CHARACTER SPACING is negative, adjacent character bodies overlap although the character symbols themselves might not.   Character spacing is specified as a fraction of the CHARACTER HEIGHT.

...en the TEXT PATH is RIGHT or LEFT, the CHARACTER SPACING is
...aled by the ratio of the length of the CHARACTER_BASE_VECTOR
... the length of the CHARACTER_UP_VECTOR.

```
CHAR_SPACING     = entity_type

    Name      : Optional string;
    SPACE     : Float;
    End;
```

Default value is 0.0

Figure 6. CHARACTER HEIGHT and CHARACTER EXPANSION FACTOR.

4.2.2 CHARACTER EXPANSION FACTOR -

The CHARACTER EXPANSION FACTOR specifies the deviation of the width to height ratio of the character from the ratio indicated by the font designer (see Figure 6). The character expansion factor is a scalar. The resulting character width is the product of the CHARACTER HEIGHT multiplied by the width/height ratio (a characteristic of each font and a quantity that can vary on a character-by-character basis) for the character multiplied by the CHARACTER EXPANSION FACTOR. The character width so derived is further scaled by multiplying it by the ratio of the length of the CHARACTER_BASE_VECTOR to the length

CHAR_ORIENT   = entity_type

Name   : Optional string;
C_UP   : Pointer (Vector);
C_BASE   : Pointer (Vector);
End;

Default value is (0.1,0,1.0,0). (C_UP and C_BASE must not be parallel).
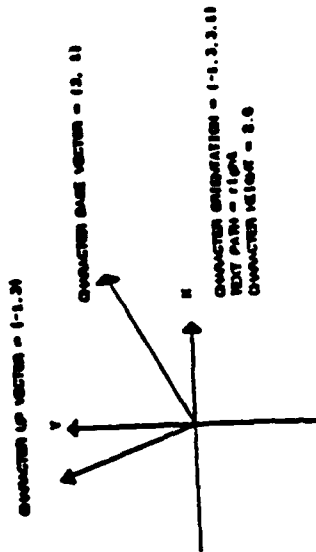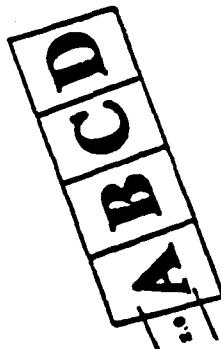
Figure 8.   CHARACTER ORIENTATION

The CHARACTER UP VECTOR gives the up direction of a character, and the CHARACTER_BASE VECTOR gives the direction of the base line of a character. Only the direction, not the length, of
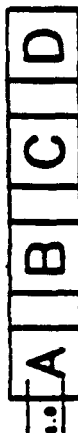
---

Figure 7. CHARACTER SPACING

4.2.4   CHARACTER ORIENTATION

CHARACTER ORIENTATION specifies the CHARACTER UP VECTOR and the CHARACTER BASE VECTOR, which fix the orientation, skew, and distortion of the characters, and also determine the sense of RIGHT, LEFT, UP, and DOWN for TEXT PATH and TEXT ALIGNMENT (see Figure 8).

these vectors is relevant. However, the ratio of the length of the base vector to the length of the up vector is used to scale the CHARACTER SPACING (for text paths RIGHT, and LEFT, and the CHARACTER EXPANSION FACTOR in all cases, before these are used to display the text.

Since the values of CHARACTER HEIGHT, CHARACTER UP VECTOR, and CHARACTER BASE_VECTOR are given in modeling coordinate units, but the characters are generated on the workstation in device coordinates using the workstation dependent font and precision, these attributes have to be transformed in such a way that the workstation can generate the characters in the intended manner.

The following is an outline of how this can be achieved, solely for the purpose of clarification. Together with the text coding, a height vector parallel to the CHARACTER UP_VECTOR with length equal to CHARACTER HEIGHT and a width vector in the direction of the CHARACTER_BASE_VECTOR with the same length as the height vector, are passed down the viewing pipeline. These vectors are transformed by the modeling transformation, viewing transformation and then by the workstation transformation in effect. Then the vectors can be used by the workstation character generator.

The height and the direction of the CHARACTER UP VECTOR, on the workstation, are given by the transformed height vector. The BASELINE direction is given by the direction of the transformed width vector. The character width is scaled from the font coordinate system to device coordinates by the length of the transformed width vector and multiplied by the CHARACTER EXPANSION FACTOR. The characters are arranged together in a text extent rectangle, depending on the values of TEXT PATH and CHARACTER SPACING. The text extent rectangle is then positioned according to the value of TEXT ALIGNMENT and the text position, contained in the definition of the TEXT primitive.

If an anisotropic transformation is in effect, the character height must be respecified for each change in orientation (see Figure 9). The CHARACTER HEIGHT and CHARACTER ORIENTATION are decoupled to permit changing character height (but not orientation) within a string. Thus, the absolute lengths of the vectors in CHARACTER ORIENTATION are not significant; only their directions and the ratio of their lengths.
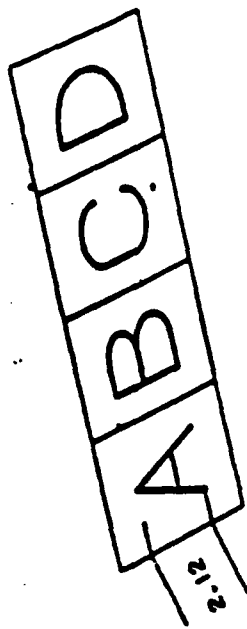
CHARACTER UP VECTOR = (-1.5,3)

CHARACTER BASE VECTOR = (4.5,1)
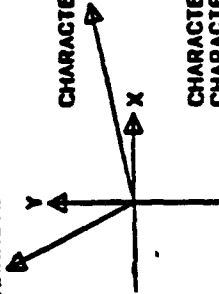
CHARACTER ORIENTATION = (-1.5,3,4.5,1)
CHARACTER PATH = right
CHARACTER HEIGHT = 2.12

Figure 9.   CHARACTER HEIGHT and CHARACTER ORIENTATION after Anisotropic Transformation.

4.2.5 TEXT PATH -

TEXT PATH has the possible values RIGHT, LEFT, UP and DOWN. It specifies the writing direction of the text string.

RIGHT means in the direction of the CHARACTER BASE VECTOR.
LEFT   means 180-degrees from the CHARACTER BASE VECTOR.
UP   means in the direction of the CHARACTER_UP VECTOR.
DOWN   means 180-degrees from the CHARACTER_UP_VECTOR.

For the UP and DOWN character path directions the characters are arranged so that the centers of the character bodies are on a straight line in the direction of the CHARACTER UP VECTOR. For LEFT and RIGHT text path directions, the characters are arranged so that the BASELINE of the characters are on a straight line

TEXT ALIGNMENT = (center, cap, 0, 0)

TEXT PATH = right
CHARACTER HEIGHT = 2.0
String = "Big"
CHARACTER HEIGHT = 1.0
Appended String = "Little"

TEXT ALIGNMENT = (right, half, 0, 0)

CHARACTER HEIGHT = 1.0
CHARACTER EXPANSION FACTOR = 1.0
String = "Normal"
CHARACTER EXPANSION FACTOR = 2.0
Appended String = "Wide"

Figure 10. Discrete Text Alignment with Appended Text and
            Proportional Spacing

If the value of TEXT ALIGNMENT is CONTINUOUS_HORIZONTAL, an
additional value, HORIZONTAL ALIGNMENT (a real number normalized
so that 1.0 corresponds to the width of the text extent
rectangle) is used as an offset from the text position to the
leftside of the text extent rectangle (see Figure 11).

If the value of TEXT ALIGNMENT is CONTINUOUS_VERTICAL, an
additional value, VERTICAL ALIGNMENT (a real number normalized
so that 1.0 corresponds to the height of the text extent
rectangle) is used as an offset from the text position to the
bottomside of the text extent rectangle.

parallel to the direction of the CHARACTER BASE_VECTOR. These
composition rules also hold true when characters of different
heights, expansion factors, or fonts are intermixed in a string
by means of attribute changes between NOT_FINAL TEXT entities
and subsequent APPEND_TEXT entities.

TEXT_PATH   = entity_type

    Name   : Optional string;
    T_PATH : String;
    End;

Default value is RIGHT

4.2.6 TEXT ALIGNMENT -

TEXT ALIGNMENT is defined for both the horizontal and vertical
directions. Horizontal alignment has the possible values:
LEFT,   CENTER,   RIGHT,   NORMAL_HORIZONTAL,   and
CONTINUOUS HORIZONTAL. Vertical alignment has the possible
values: TOP, CAP, HALF, BASE, BOTTOM, NORMAL_VERTICAL, and
CONTINUOUS_VERTICAL.

For TEXT PATH = LEFT or RIGHT,

```
TOPLINE:     topline farthest from the BASELINE.
CAPLINE:     CAPLINE farthest from the BASELINE.
HALFLINE:    halfline farthest from the BASELINE.
BOTTOMLINE:  bottomline farthest from the BASELINE.
LEFT:        leftmost edge of leftmost character body.
RIGHT:       rightmost edge of rightmost character body.
CENTER:      halfway between left and right edges.
```

For TEXT PATH = UP or DOWN,

```
TOPLINE:     topline of topmost character.
CAPLINE:     CAPLINE of topmost character.
HALFLINE:    halfway between halflines of topmost and bottommost
             character.
BASELINE:    BASELINE of bottommost character.
BOTTOMLINE:  bottomline of bottommost character.
LEFT:        left edge farthest from the center line.
RIGHT:       right edge farthest from the center line.
CENTER:      on centerline.
```

The 'normal' parameters are dependent on the text path at the time of the elaboration of the text elements.

PATH	NORMAL_HORIZONTAL	NORMAL_VERTICAL
RIGHT	LEFT	BASELINE
LEFT	RIGHT	BASELINE
UP	CENTER	BASELINE
DOWN	CENTER	TOP

```
TEXT_ALIGN    = entity_type

    Name    : Optional string;
    H_ALIGN : String;
    V_ALIGN : String;
    CONT_H  : Optional Float;
    CONT_V  : Optional Float;
    End;
```

Default value is (NORMAL_HORIZONTAL, NORMAL_VERTICAL)

4.2.7 Discussion -

The foregoing examples have been illustrated for the case of the CHARACTER_UP_VECTOR and the CHARACTER_BASE_VECTOR being orthogonal. When they are not, the text extent rectangle becomes a parallelogram with the sides remaining parallel to the two orientation vectors. The centerline skews to remain parallel with the left and right edges of the text extent

---

```
TEXT ALIGNMENT = (continuous
    horizontal,base,0.25,0)
CHARACTER PATH = right
```

```
TEXT ALIGNMENT = (continuous
    horizontal,continuous vertical,
    -0.25,-0.25)
CHARACTER PATH = right
```

```
String 1 = ABCD
TEXT ALIGNMENT = (left,continuous
    vertical,0,1)
CHARACTER PATH = right
```

```
String 2 = EFGH
TEXT ALIGNMENT = (left,continuous
    vertical,0,2.5)
CHARACTER PATH = right
```

```
String 1 = ABCD
TEXT ALIGNMENT = (continuous
    horizontal,top,0,0)
CHARACTER PATH = down
```

```
String 2 = EFGH
TEXT ALIGNMENT = (continuous
    horizontal,top,2,0,0)
CHARACTER PATH = down
```

Figure 11. Continuous Text Alignment

Alignment of text is done with respect to a text extent rectangle which is derived by joining the character bodies of the characters in the string according to the current status of the attributes and the composition rules described.

CHARACTER UP VECTOR = (-1,5,3)

CHARACTER BASE VECTOR = (4,5,1)

CHARACTER ORIENTATION = (-1,5,3,4,5,1)
CHARACTER PATH = right
CHARACTER HEIGHT = 2.12

TEXT ALIGNMENT = (left,continuous vertical, 0,2.0)

Figure 12.   Continuous Text Alignment after Anisotropic Transformation

As an example of the set of the continuous alignment attributes, consider the display of four rows of left-justified text, each consisting of a single string specified by a single TEXT element. To ensure that ascenders and descenders do not interfere between rows and that, in addition, there is a space of at least one-half the maximum size of a character between the descenders of one row with raised accent marks or oversized symbols of another row, TEXT ALIGNMENT should be set to (LEFT, CONTINUOUS_VERTICAL), and VERTICAL ALIGNMENT to 0.0. Then, output the first row with the text position equal to the lower-left corner of the string. Now, set VERTICAL ALIGNMENT to 1.5, and output the second string with the same text position. This places the second row below the first because of the change in alignment. The last two rows are output in the same way with

---

parallelogram. The height of the text extent rectangle is measured along the skewed edge (not perpendicular to the BASELINE), and the distance to be moved for alignment is done along the angle made by the appropriate orientation vector. RIGHT is in the direction of the CHARACTER_BASE_VECTOR, and LEFT is in the opposite direction.

The continuous values of alignment are necessary for displaying multiline text, to ensure that the ascenders of one row and the descenders of the next do not overlap. Both continuous horizontal and continuous vertical alignments are provided in order to provide this ability for all values of TEXT PATH. Since inquiry of the dimensions of the text extent rectangle cannot be provided in a blind interchange environment, other means have been provided to align more than one row of characters to the same text position. This means that the alignment parameters must be able to exceed the maximum dimensions of the text extent rectangle. Use of a continuous parameter allows specification of interrow space, analogous to intercharacter space (see Figure 12).

It should be recognized that STROKE precision does not necessarily mean vector strokes; as long as the representation adheres to the rules governing STROKE precision, the font may be realized in any form, (for example by raster fonts).

4.2.9 TEXT FONT INDEX -

The TEXT FONT INDEX attribute is used to select a particular font on the workstation. Every workstation must support at least on font that is able to generate a graphical representation of characters defined in ISO 646. This shall be font index number 1).

4.2.10 TEXT COLOR INDEX -

The TEXT COLOR INDEX attribute is a pointer into the color table. Subsequent text entities are to be displayed with this color.

4.3 Text Display

4.3.1 TEXT -

The character codes specified in the string are interpreted to obtain the associated symbols from the currently selected font. Characters are displayed on the view surface as specified by the text attributes.

The FLAG parameter is used to permit changing the following text attributes within a string which will be aligned as a single block: TEXT FONT INDEX, TEXT PRECISION, CHARACTER EXPANSION FACTOR, CHARACTER SPACING, TEXT COLOR INDEX, CHARACTER HEIGHT, and current TEXT_ATTR entity.

If the FLAG is set to NOT_FINAL, the character codes in the string parameter are accumulated, along with the current attribute settings. In this case, only the setting of attributes listed above is allowed between this entity and the APPEND_TEXT entity.

If the FLAG is set to FINAL, the string parameter constitutes the entire string to be displayed. The position of the string relative to the text reference point parameter is according to TEXT ALIGNMENT. Text entities with a null string parameter are legal and may be followed by the allowed text attributes and APPEND_TEXT as described above.

VERTICAL ALIGNMENT set to 1.0 and 4.5, and the text position parameters to TEXT unchanged. A value of 1.0 assures no overlap between rows; anything greater than 1.0 guarantees additional non-printing space. A similar use of the continuous vertical parameter can be made when the text path is DOWN.

The TEXT attributes apply in the same way to the RESTRICTED TEXT primitive. Because determination of the extent of a string is generally not possible in a blind interchange environment, the RESTRICTED TEXT element has as a parameter the size of a text extent region. If the text string, as displayed by the current text attributes, would exceed the parallelogram derived from this parameter, then the text attributes CHARACTER EXPANSION FACTOR, CHARACTER SPACING, TEXT FONT INDEX, TEXT PRECISION, and CHARACTER HEIGHT are adjusted in an implementation-dependent manner so that all of the specified text string fits within the parallelogram and the extent of the displayed string does not exceed the parallelogram.

4.2.8 TEXT PRECISION -

The TEXT PRECISION attribute is used to select the 'closeness' of the text rendition on the display surface to that defined by the workstation independent text attributes and the applicable transformation/clipping. The text precision attribute has the following possible values: STRING, CHAR, and, STROKE.

1. STRING: Only the starting position of subsequent text strings need be guaranteed, and the manner in which the string is clipped is implementation dependent.

2. CHAR: The implementation must guarantee that the starting position of each character satisfy the relevant text attributes, thus guaranteeing orientation and placement of the string; however, skew, orientation, and size of each character are not guaranteed. All characters on the string which lie completely inside or outside the clipping region are clipped as appropriate, but the effect of clipping on a character whose character box is intersected by the clipping boundary is implementation dependent.

3. STROKE: The implementation must guarantee that the placement, skew, orientation, and size of all characters satisfy all standardized text attributes. Characters are clipped to the geometric accuracy of the device.

TEXT_PRECISION = entity_type

```
Name     : Optional string;
TP       : String;
End;
```

Default value is STRING

TEXT   = entity_type

```
    Name   : Optional string;
    REF_PT : Pointer (Point3); String;
    FLAG   : FINAL or NOT_FINAL;
    T_STR  : String;
    End;
```

4.3.2 RESTRICTED_TEXT -

RESTRICTED_TEXT behaves as does TEXT, with the exception that the text is constrained to be within a parallelogram determined by the 'extent' parameters, the position, and the text attributes.

The first component of the 'extent' parameter is measured parallel to the base vector of CHARACTER ORIENTATION, and the second component is measured parallel to the up vector. A parallelogram is formed whose sides are parallel to the vectors, and which have lengths as in the 'extent' parameter. The parallelogram is placed at the reference point and aligned as per the current TEXT ALIGNMENT.

All text in the string is displayed within the resulting 'extent' parallelogram. The text attributes CHARACTER HEIGHT, CHARACTER EXPANSION FACTOR, CHARACTER SPACING, TEXT PRECISION, AND TEXT FONT INDEX are varied if necessary, and only if necessary, to achieve the extent restriction. The method of varying the attributes is implementation dependent.

The FLAG parameter is used as it is in TEXT. Appended text, if any, must also be set to FINAL or NOT_FINAL. fit within the prescribed 'extent' parallelogram.

RESTRICTED_TEXT = entity_type

```
    Name    : Optional string;
    EXT_WID : Float;
    EXT_HI  : Float;
    REF_PT  : Pointer (Point3); String;
    FLAG    : FINAL or NOT_FINAL;
    T_STR   : String;
    End;
```

4.3.3 APPEND_TEXT -

The character codes specified in the string are appended to the string defined by preceding NOT_FINAL TEXT, RESTRICTED TEXT, and APPEND_TEXT entities. The codes are interpreted to obtain the associated symbols from the current text font.

The FLAG parameter is used as it is in TEXT, and may similarly be set to FINAL or NOT_FINAL.

APPEND_TEXT   = entity_type

```
    Name    : Optional string;
    FLAG    : FINAL or NOT_FINAL;
    T_STR   : String;
    End;
```

4.3.4 TEXT_ATTR -

TEXT_ATTR   = entity_type

```
    Name    : Optional String;
    C_HT     : Optional Pointer (CHAR_HEIGHT);
    C_EXPAN  : Optional Pointer (CHAR_EXPAN);
    C_SPAC   : Optional Pointer(CHAR_SPACING);
    C_ORINT  : Optional Pointer (CHAR_ORIENT);
    T_PATH   : Optional Pointer (TEXT_PATH);
    T_ALIGN  : Optional Pointer (TEXT_ALIGN);
    T_PREC   : Optional Pointer (TEXT_PRECISION);
    FONT_IND : Optional Integer;
    COL_TAB  : Optional Pointer (COLOR_TABLE);
    COL_IND  : Optional Integer;
    End;
```

4.4 Line Attributes

LINE_ATTR   = entity_type

```
    Name    : Optional String;
    LIN_WID : Optional Float - width scale factor;
    LIN_TAB : Optional Pointer (LINE_TYPE_TABLE);
    LIN_IND : Optional Integer - line index;
    COL_TAB : Optional Pointer (COLOR_TABLE);
    COL_IND : Optional Integer - color index;
    End;
```

; Default values: LIN_WID = 1

Line width is the nominal line width of the workstation multiplied by the LIN_WID scale factor. If LIN_WID .le. 0, then the line width is the minimum supported by the workstation.

Default color table:

```
BAK_COL points to        0 0 0    Background = Black
FOR_LEN = 8
IND_1 points to          0 0 0    Black
IND_2 points to          1 0 0    Red
IND_3 points to          0 1 0    Green
IND_4 points to          0 0 1    Blue
IND_5 points to          1 1 0    Yellow
IND_6 points to          1 0 1    Magenta
IND_7 points to          0 1 1    Cyan
IND_8 points to          1 1 1    White
```

4.9 Color Attributes

COLOR_ATTR   = entity_type

```
Name   : Optional String;
RED    : Float; Amount of Red [0-1]
GREEN  : Float; Amount of Green [0-1]
BLUE   : Float; Amount of Blue [0-1]
End;
```

4.10 Edge Attributes

EDGE_ATTR   = entity_type

```
Name    : Optional String;
E_FLAG  : Optional String; ON or OFF
COL_TAB : Optional Pointer (COLOR_TABLE);
COL_IND : Optional Integer; edge color index.
TYP_TAB : Optional Pointer (LINE_TYPE_TABLE);
TYP_IND : Optional Integer; edge type index.
EDG_WID : Optional Float; Edge width scale factor.
End;
```

The EDGE FLAG defines when edges should be displayed. If this flag is ON, visible fill area edges will be displayed according to the edge type specified by TYP_IND, line width scale factor defined by EDG_WID, and color specified by COL_IND.

4.11 Interior Attributes

INTERIOR_ATTR   = entity_type

```
Name    : Optional String;
STYLE   : Optional String;
STL_TAB : Optional Pointer (PATTERN TABLE or
                           CROSS_HATCH_TABLE);
STL_IND : Optional Integer; Style index
COL_TAB : Optional Pointer (COLOR_TABLE);
COL_IND : Optional Index; color index.
End;
```

4.5 Surface Attributes

SURFACE_ATTR   = entity_type

```
Name    : Optional String;
EDG_ATR : Optional Pointer (EDGE_ATTR);
INT_ATR : Optional Pointer (INTERIOR_ATTR);
End;
```

4.6 Line Type Table

LINE_TYPE_TABLE = entity_type

```
Name    : Optional String;
TAB_LEN : Integer;
FONT_1  : Pointer (LINE_FONT);
     .
     .
FONT_N  : Pointer (LINE_FONT);
End;
```

Default values for TAB_LEN .le. 0:

```
1 - solid line
2 - dashed line
3 - dotted line
4 - dashed-dotted line
```

4.7 Line Font

LINE_FONT   = entity_type

```
Name  : Optional String;
LEN   : Integer; Number of bits in font pattern.
LINE  : Integer array; the repeated bit pattern.
End;
```

4.8 Color Table

COLOR_TABLE   = entity_type

```
Name    : Optional String;
BAK_COL : Pointer (COLOR_ATTR); Background color.
FOR_LEN : Integer; Number of foreground colors.
IND_1   : Pointer (COLOR_ATTR); Foreground color 1.
     .
     .
IND_N   : Pointer (COLOR_ATTR); Foreground color N.
End;
```

The Interior Style (STYLE) is used to determine in what style the interior area should be filled. It has one of the following values:

1. HOLLOW: No filling, but draw the bounding polyline, using the interior color index currently selected. The linetype and linewidth of this bounding polyline are implementation dependent.

2. SOLID: Fill the interior of the area using the interior COL_IND currently in use.

3. PATTERN: Fill the interior of the area using interior STL_IND currently in use as an index into the PATTERN_TABLE.

4. HATCH: Fill the interior of the area using interior STL_IND currently in use as an index into the CROSS_HATCH_TABLE.

5. EMPTY: No filling. Areas drawn using this value will be invisible.

4.12 Pattern Table

```
PATTERN_TABLE   = entity_type
    Name    : Optional String;
    TAB_LEN : Integer;
    IND_1   : Pointer (PATTERN_ATTR);
    .
    .
    IND_N   : Pointer (PATTERN_ATTR);
    End;
```

4.13 Pattern Attributes

```
PATTERN_ATTR   = entity_type
    Name : Optional String;     Pattern Size
    SIZE : Float;
    PRP  : Pointer (Point3);    Pattern Ref Point
    PPV  : Pointer (Vector3);   Pattern Plane Vector
    PUV  : Pointer (Vector3);   Pattern Up Vector
    End;
```

For interior style PATTERN, the pattern is defined by the pattern representation. A grid of cells is defined. The color is specified individually for each cell by a color index, which is a pointer into the color table. The size and position of the start of the pattern are defined by a workstation independent PATTERN_ATTR entity. The attributes are subject to all transformations producing a transformed pattern parallelogram. The pattern is mapped onto the fill area by conceptually replicating the transformed pattern parallelogram in directions

parallel to its sides until the interior of the complete fill area is covered.

The pattern parallelogram is defined as follows:

1. The pattern plane is defined by the pattern reference point, pattern up vector, and pattern plane vector.

2. This pattern plane is shifted along the fill area plane normal such that the pattern reference point is in the fill area plane.

3. The shifted pattern plane now intersects the plane of the fill area in a line (unless the planes are coincident). The shifted pattern reference point is in this line. This point becomes the origin for patterning.

4. The shifted pattern is rotated around the intersecting line of the two planes, over the smallest angle, until the pattern plane coincides with the polygon plane. The shifted and rotated pattern plane vector and pattern up vector now lie in the fill area plane.

5. Three vertices defining the pattern parallelogram are finally defined in the pattern plane. One vertex is the shifted pattern reference point. A second vertex is derived by measuring the distance SX (pattern size) from the pattern origin along the pattern up vector. A third vertex is derived by measuring the distance SY (pattern size) from the pattern origin along the pattern plane vector.

Mapping the transformed pattern cells to the pixels of a raster display is performed by the following rules:

1. If the center of a pixel lies inside the parallelogram defined by the transformed cell, its color is set:

2. The pixel will be assigned the color of the cell corresponding to the pixel's center.

If a workstation is not capable of representing the interior style SOLID, PATTERN or HATCH, the minimal action required is the interior style HOLLOW.

4.14 Cross Hatch Table

```
CROSS_HATCH_TABLE        = entity_type

    Name    : Optional String;
    TAB_LEN : Integer;
    IND_1   : Pointer (CROSS_HATCH_ATTR);

    IND_N   : Pointer (CROSS_HATCH_ATTR);
    End;
```

4.15 Cross Hatch Attributes
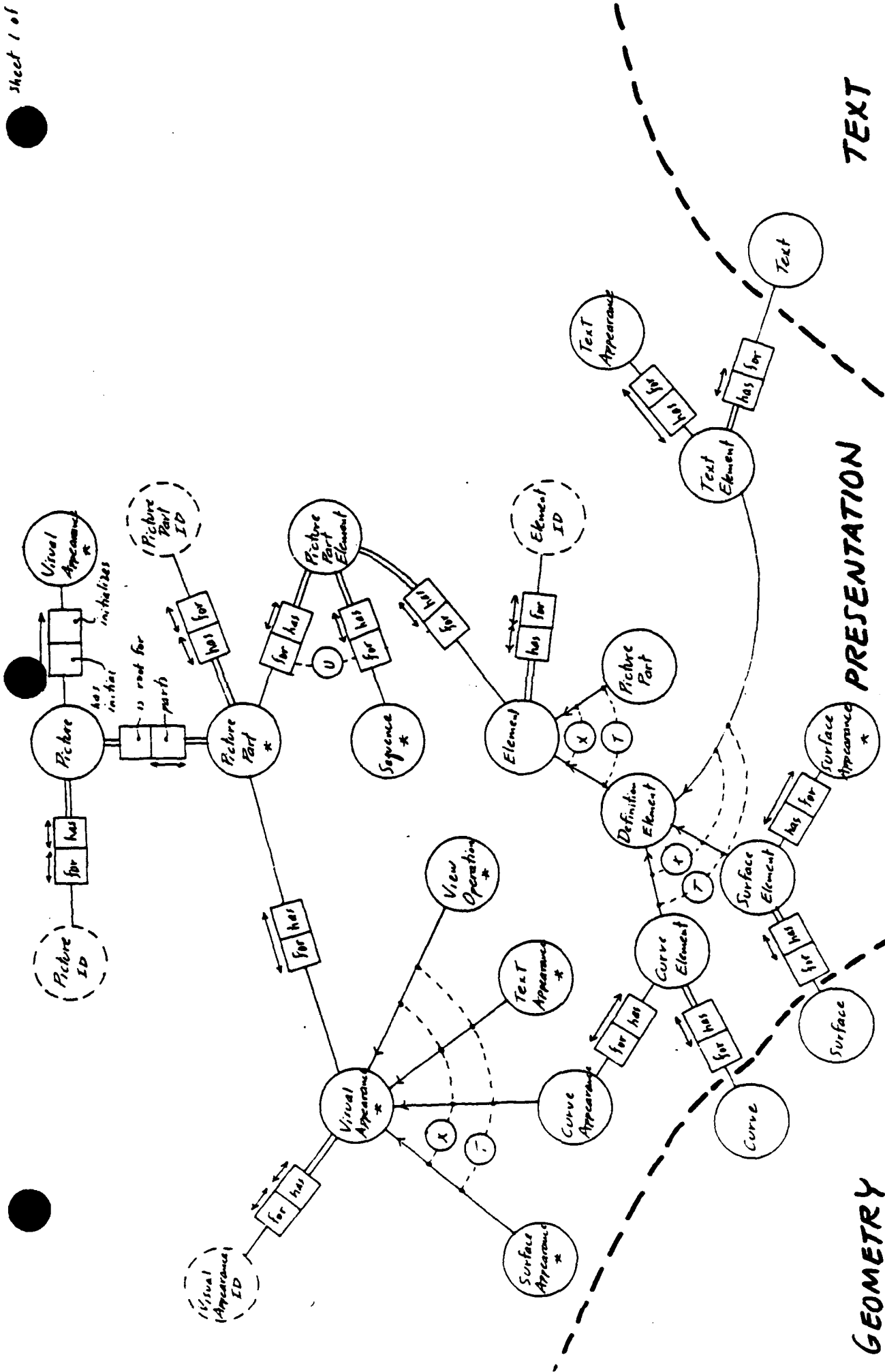
```
CROSS_HATCH_ATTR         = entity_type

    (TBS)
```

5 REFERENCES

1. American National Standard for the Functional Specification of the Programmer's Hierarchical Interactive Graphics Standard (PHIGS). X3H3, Revised Feb 18, 1985.

2. Graphical Kernel System for Three Dimensions (GKS-3D) (Draft). ISO/TC97/SC21/WG5-2 N277, Revised Jan 10, 1985.

3. "Status Report of the Graphics Standards Planning Committee of ACM/SIGGRAPH" (GSPC) Computer Graphics, Vol 13, No. 3, August 1979.

4. "CG-VDI Baseline Document." (Computer Graphics Virtual Device Interface). Part 1, Functional Specification, ISO/TC97 N-1511, ANSI X3H3/85-07, ANSI X3H3J/86-10R2, Revision 3, March 1985.

5. Foley, J.D., and A. Van Dam, "Fundamentals of Interactive Computer Graphics," Addison-Wesley, Reading, Mass., 1982.

6. Newman, W.M., and R.F. Sproull, "Principles of Interactive Computer Graphics," Second Edition, McGraw Hill, New York, 1979.

Figure 11. Mapping of Cell Arrays

For a workstation which can implement patterns but not transformable patterns, a suitable action is to generate non-transformed patterns to fill a fill area.

For interior style HATCH, the cross hatch index selects from a set of predefined workstation-dependent hatch patterns. The minimal required support for cross-hatch patterns is 1, which may be identical to HOLLOW.

Appendix C2.2

Task 1

Presentation Model (NIAM)

(RESOURCE)

PRESENTATION

TEXT

GEOMETRY

CURVE
APPEARANCE

Empty

Hatch

Hollow

Solid

Pattern

Interior Style Definition

Interior Style Keyword

empty
hatch
pattern
solid
hollow

has for

Color *

Interior Color Index

has for

has for

Interior Color

Surface Appearance Type

Surface Appearance Type Code

has for

has for

Interior Appearance

Surface Appearance

Interior Style

say def

Interior Style Index

Edge Width

say def

say def

Edge Width Index

Magnitude *

Edge Appearance

Edge Color

say def

Color *

has for

Edge Color Index

Edge Font

defines by

defines

has for

Curve Font *

Edge Font Index

SURFACE
APPEARANCE

TEXT APPEARANCE

GEOMETRY
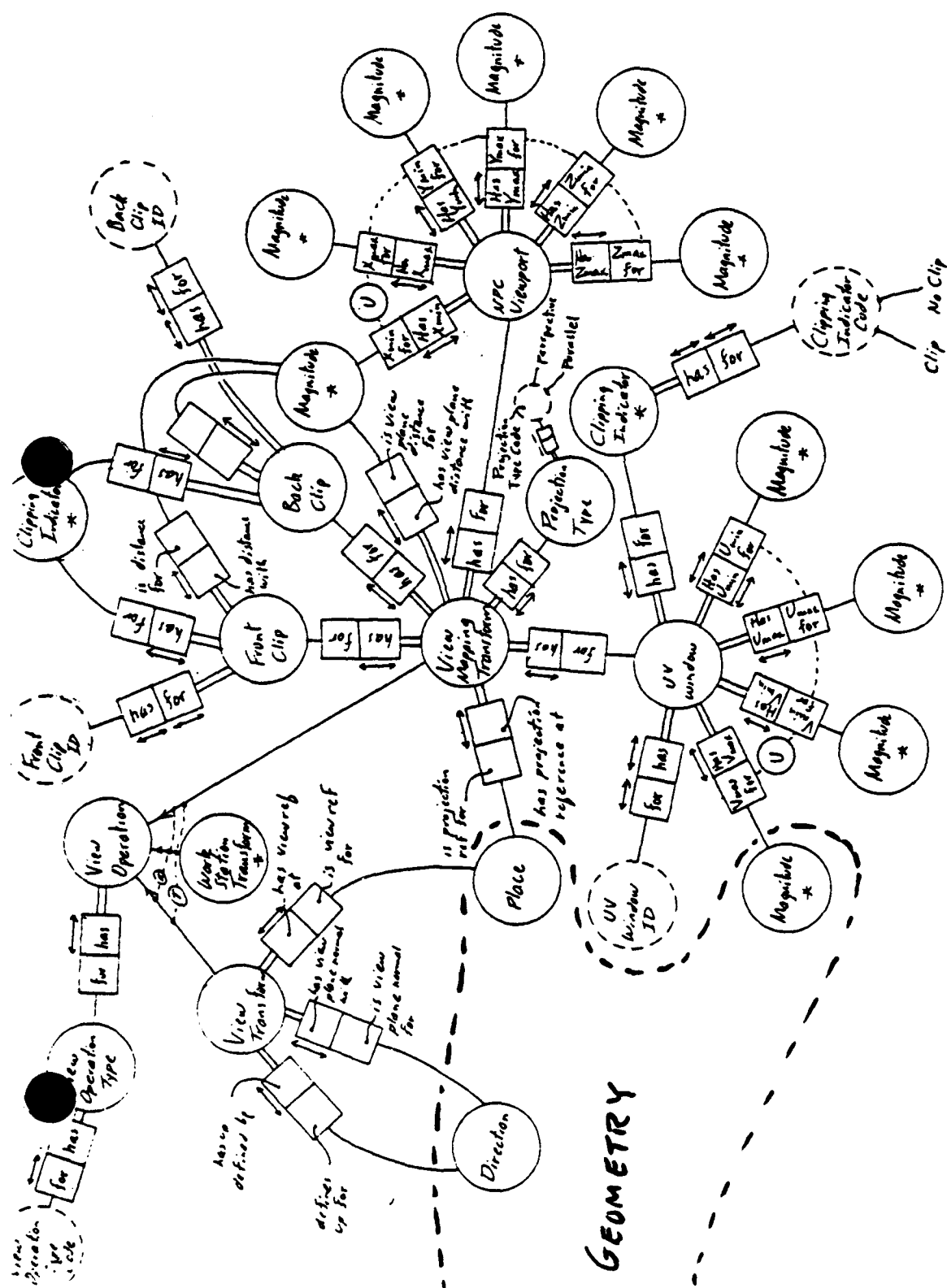
VIEW OPERATION

TEXT

GEOM
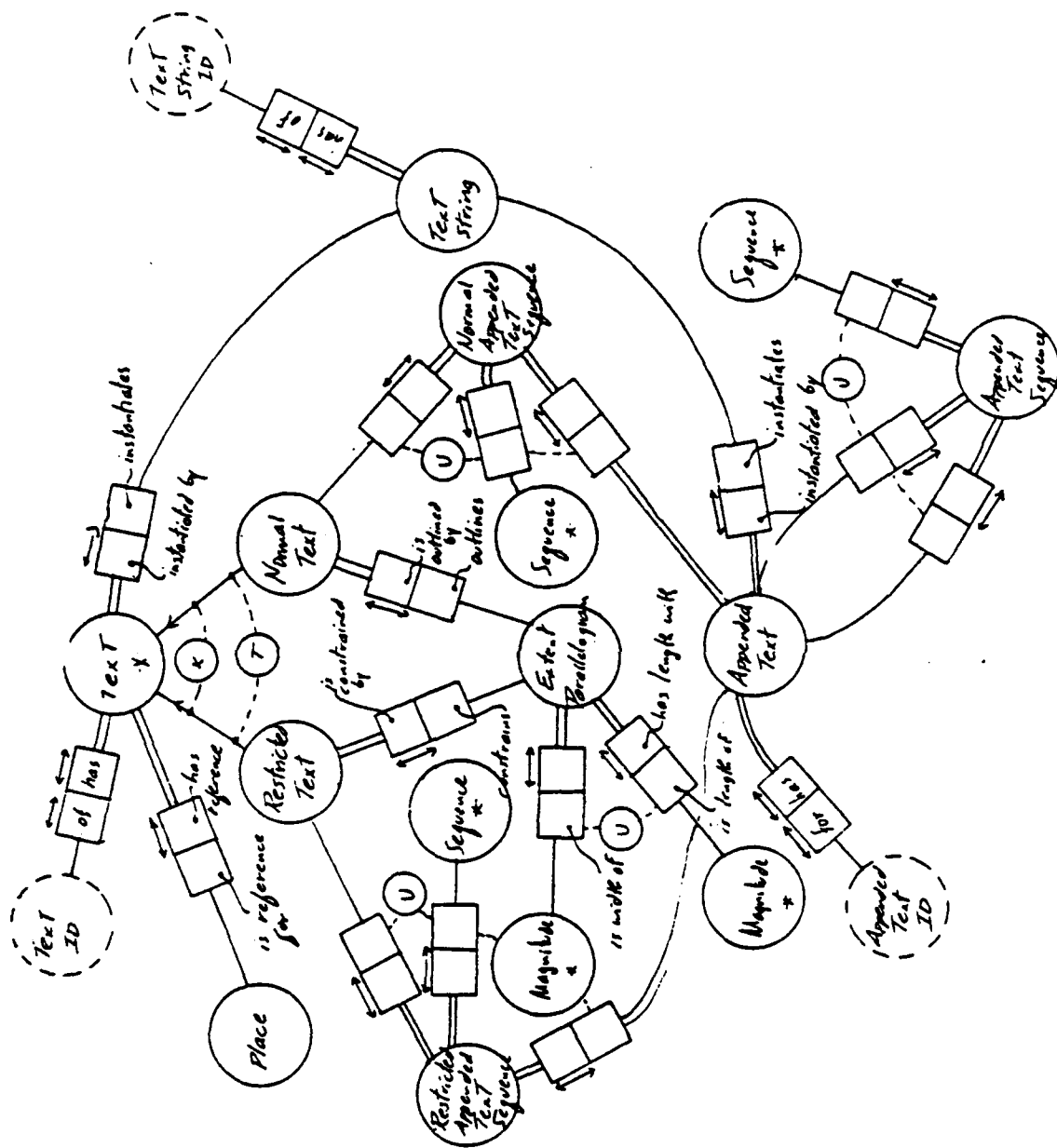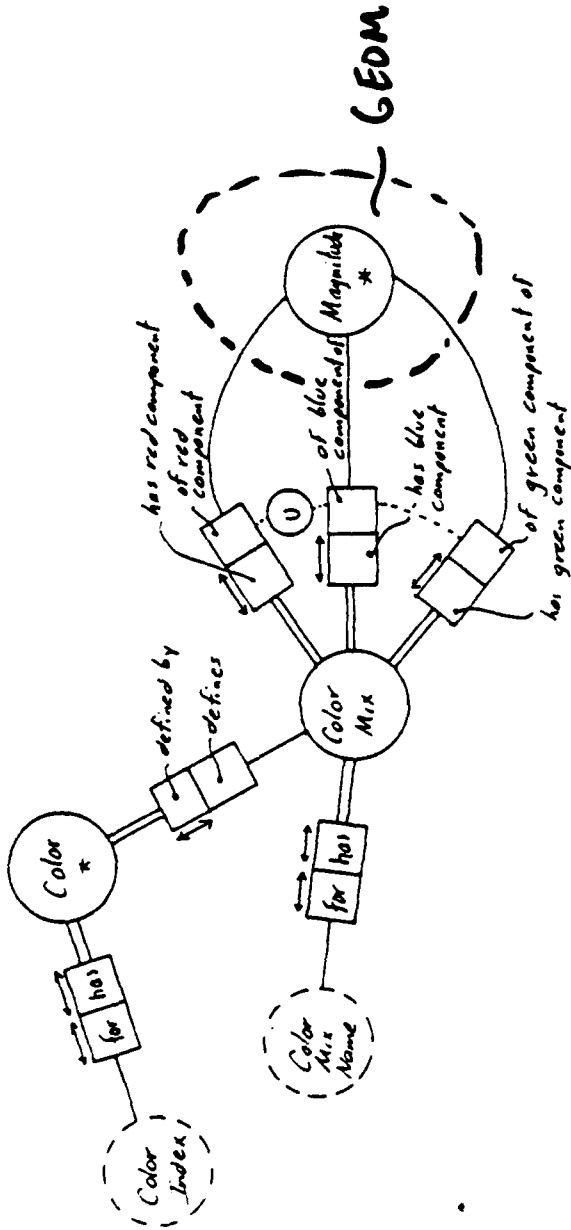
COLOR
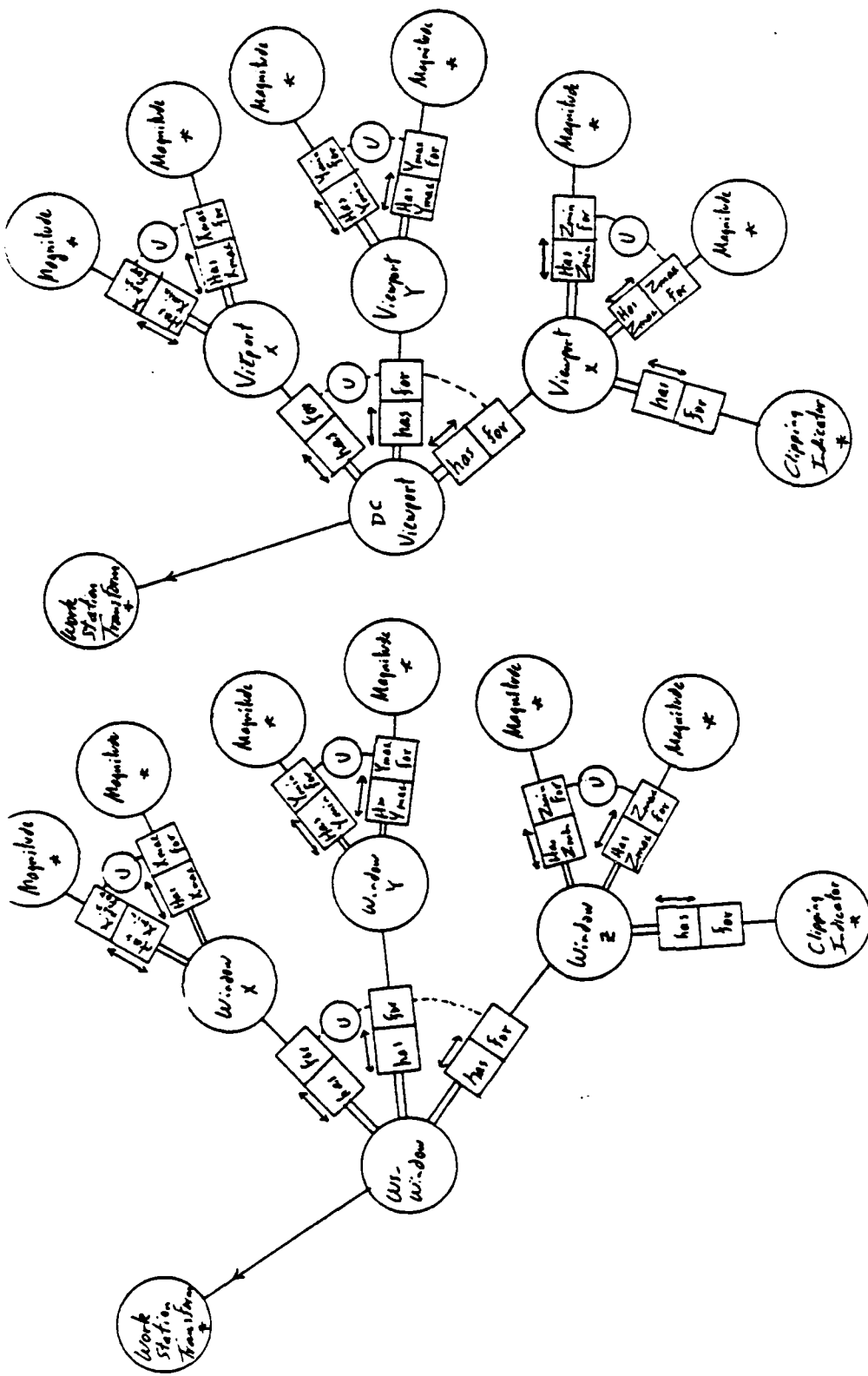
Work Station Transforms

Appendix C2.3

Task 1

Presentation Model (DSL)

(RESOURCE)

```
-- START OF PRESENTATION ENTITIES
-- VERSION 0.1
-- AUTHOR  WINFREY

ENTITY picture;
NOROLE
  name : OPTIONAL t_name WHERE UNIQUE;
ROLE
  initial_visual_appearance : SET OF REFER(visual_appearance);
  picture_part : DEPENDENT SET OF REFER(picture_part);
END;


ENTITY picture_part;
NOROLE
  name : OPTIONAL t_name WHERE UNIQUE;
ROLE
  picture_part_appearance : SET OF REFER(visual_appearance);
  element : LIST(0 TO *) OF REFER(picture_element);
END;


ENTITY curve_font;
NOROLE
  name : OPTIONAL t_name WHERE UNIQUE;
ROLE
  pattern : REFER(curve_font_bit_pattern);
END;


ENTITY curve_font_bit_pattern;
NOROLE
  name : OPTIONAL t_name WHERE UNIQUE;
ROLE
  num_of_bits : INTEGER;
  bit_pattern : ARRAY(1:num_of_bits) OF t_bit;
END;


ENTITY curve_width;
NOROLE
  name : OPTIONAL t_name WHERE UNIQUE;
ROLE
  width : t_magnitude;
END;


ENTITY curve_color;
NOROLE
  name : OPTIONAL t_name WHERE UNIQUE;
ROLE
  color_value : REFER(color);
END;
```

```
ENTITY edge_font;
NOROLE
  name : OPTIONAL t_name WHERE UNIQUE;
ROLE
  edge_font_pattern : REFER(curve_font);
END;


ENTITY edge_color;
NOROLE
  name : OPTIONAL t_name WHERE UNIQUE;
ROLE
  color_value : REFER(color);
END;


ENTITY edge_width;
NOROLE
  name : OPTIONAL t_name WHERE UNIQUE;
ROLE
  width : t_magnitude;
END;


ENTITY interior_style;
NOROLE
  name : OPTIONAL t_name WHERE UNIQUE;
ROLE
  style_definition : REFER(interior_style_definition);
END;


ENTITY interior_color;
NOROLE
  name : OPTIONAL t_name WHERE UNIQUE;
ROLE
  color_value : REFER(color);
END;


ENTITY uv_window;
NOROLE
  name : OPTIONAL t_name WHERE UNIQUE;
ROLE
  clip : t_clip_indicator;
  u_min, u_max,
  v_min, v_max : t_magnitude;
END;
```

```
ENTITY npc_viewport;
NOROLE
  name : OPTIONAL t_name WHERE UNIQUE;
ROLE
  x_min, x_max,
  y_min, y_max,
  z_min, z_max : t_magnitude;
END;


ENTITY view_mapping_transform;
NOROLE
  name : OPTIONAL t_name WHERE UNIQUE;
ROLE
  clip_front : t_magnitude;
  clip_back : t_magnitude;
  view_plane_distance : t_magnitude;
  uv_window : REFER(uv_window);
  npc_viewport : REFER(npc_viewport);
  projection_type : t_projection_type_code;
  projection_reference : REFER(place);
END;


ENTITY view_transform;
NOROLE
  name : OPTIONAL t_name WHERE UNIQUE;
ROLE
  up : REFER(direction);
  view_plane_normal : REFER(direction);
  view_reference : REFER(place);
END;


ENTITY color;
NOROLE
  name : OPTIONAL t_name WHERE UNIQUE;
ROLE
  red : t_magnitude;
  blue : t_magnitude;
  green : t_magnitude;
END;


ENTITY restricted_text;
NOROLE
  name : OPTIONAL t_name WHERE UNIQUE;
ROLE
  location : REFER(place);
  basic_text : STRING;
  appended_text : LIST(0 TO *) OF STRING;
  outline : t_parallelogram;
END;
```

11

```
ENTITY normal_text;
NOROLE
   name : OPTIONAL t_name WHERE UNIQUE;
ROLE
   location : REFER(place);
   basic_text : STRING;
   appended_text : LIST(O TO *) OF STRING;
   outline : t_parallelogram;
END;


ENTITY pattern; -- Note(1)
NOROLE
   name : OPTIONAL t_name WHERE UNIQUE;
END;


ENTITY solid; -- Note(1)
NOROLE
   name : OPTIONAL t_name WHERE UNIQUE;
END;


ENTITY hollow; -- Note(1)
NOROLE
   name : OPTIONAL t_name WHERE UNIQUE;
END;


ENTITY empty; -- Note(1)
NOROLE
   name : OPTIONAL t_name WHERE UNIQUE;
END;


ENTITY char_font; -- Note(1)
NOROLE
   name : OPTIONAL t_name WHERE UNIQUE;
END;


ENTITY text_color; -- Note(1)
NOROLE
   name : OPTIONAL t_name WHERE UNIQUE;
END;


ENTITY text_allignment; -- Note(1)
NOROLE
   name : OPTIONAL t_name WHERE UNIQUE;
END;
```

```
ENTITY char_expansion; -- Note(1)
NOROLE
  name : OPTIONAL t_name WHERE UNIQUE;
END;


ENTITY char_height; -- Note(1)
NOROLE
  name : OPTIONAL t_name WHERE UNIQUE;
END;


ENTITY char_spacing; -- Note(1)
NOROLE
  name : OPTIONAL t_name WHERE UNIQUE;
END;


ENTITY char_orientation; -- Note(1)
NOROLE
  name : OPTIONAL t_name WHERE UNIQUE;
END;


ENTITY text_path; -- Note(1)
NOROLE
  name : OPTIONAL t_name WHERE UNIQUE;
END;


ENTITY text_precision; -- Note(1)
NOROLE
  name : OPTIONAL t_name WHERE UNIQUE;
END;


ASSOCIATION text_element;
  OF (text, text_appearance( * ));
  -- specifying the cardinality of an association has not been
  -- settled. This tries to say that one "text" may be associated
  -- with many "text_appearance".
END;


ASSOCIATION curve_element;
  OF (curve, curve_appearance( * ));
END;


ASSOCIATION surface_element;
  OF (surface, surface_appearance( * ));
END;
```

```
CLASS picture_element OF
   (definition_element, picture_part);


CLASS visual_appearance OF
   (text_appearance, surface_appearance, curve_appearance,
    view_operation);


CLASS surface_appearance OF
   (edge_appearance, interior_appearance);


CLASS curve_appearance OF
   (curve_font, curve_color, curve_width);


CLASS text_appearance OF
   (char_font, text_color, text_allignment, char_expansion,
    char_height, char_spacing, char_orientation, text_path,
    text_precision);


CLASS view_operation OF
   (view_transform, view_mapping_transform,
    work_station_transform);


CLASS definition_element OF
   (curve_element, surface_element, text_element);


CLASS interior_appearance OF
   (interior_style, interior_color);


CLASS edge_appearance OF
   (edge_font, edge_color, edge_width);


CLASS interior_style_definition OF
   (pattern, solid, hollow, hatch, empty);


CLASS text OF
   (normal_text, restricted_text);


--    NOTE(1) These entities require ROLE attributes, but the NIAM
--            model did not specify what those ROLE attributes are.
--            This will have to be corrected after L.L. Initiation.
```

Appendix C3

Task 1

Flat Plate Mechanical Part Application Model

(DISCIPLINE)

C3.1   Initial Model and Documentation

C3.2   Scernarios for Presentation of Flat Plate with Holes

C3.3   Qualified Model (NIAM)

C3.4   Global Model (NIAM)

C3.5   Specification Model (DSL)

Appendix C3.1

Task 1

Flat Plate Mechanical Part Model Initial Model and Documentation

(DISCIPLINE)

Flat Plate Reference Model

Scope of the model

For the purposes of this initial document, flat plates are considered to have the following characteristics:

- Plate has two major (large) faces, a top and a bottom

- Uniform thickness; i.e. top and bottom faces are parallel

- The top and bottom faces are not required to have congruent perimeters

- The thickness is small realtive to the top and bottom face dimensions

- For any traverse from top to bottomface, along the intersection of an arbitrary plane perpendicular to the top and bottom faces, only one "side face" will be encountered.

- Only round, through holes of uniform diameter are allowed.

- Edge conditions, e.g. chamfer/radius specifications will be treated as attributes. If the chamfers/radii are large enough to require more detailed definition, they are outside the scope of this model.

Definitions:

Flat Plate

This is an entity that is the "top of the tree" in the definition of a single piece part. Under this entity is all the definitional information for the part.

Face

Face is a topological entity. It is essentially a bounded surface. The boundaries are topological entities, edges, with their end points defined by vertices. This is the relatively standard BREP notation.

Surface

The geometric (mathematical) entity that defines the shape of a face.

Datum

One of three planes that in principle define the coordinate system of the plate. The three planes are mutually perpendicular and intersect in a point. That point is the origin. The planes or some manifestation of them are referenced as datums in tolerances and other entities.

Angular + Location + Geometric tolerances

These tolerances are defined in the tolerance model. Their presence in this model is to indicate the relationships of tolerances to the flat plate part.

Heat Treat Specification

A specification defined by industry groups or a company that defines a specific set of heat treatment procedures and results to be applied to a part.

Drawing

A manifestation in human readable form of some of the definitional information about a part.

Process Plan

This is a grouping of all the information needed to manufacture the plate from raw material.

3D Views

These views are 2D projections of the complete 3D model definition on a viewing plane with or without hidden line and surface removal. They are used for many purposes, such as instruction sheets for the shop.

Cross section views

These views are 2D representations of a part as though a plane has cut away the portion of the part nearest the viewer. These may also be used for many purposes such as instructions, visualization or drawings.

Profile views

These views are 2D representations of a part from standard viewpoints. These might be used on a drawing or by a process planner to nest plate parts in raw material for NC burners.

Edge

The topological entity that bounds a face. It is essentially a geometric curve, bounded by end points (vertices).

Curve

The geometric (mathematical) entity that defines the shape of an edge.

Vertex

The topological entity that bounds an edge. It is essentially a geometric point.

Coordinate Point

The geometric (mathematical) entity that defines a location in space.

Hole

A feature of the flat plate. It is defined by its diameter, centerline vector and a coordinate point contained on a surface of the plate. This feature entity may be required, in some applications to be transformed into topology and geometry, e.g. an assortment of faces and edges, but for purposes of model definition, this one is informationally complete within the scope of the model.

Surface texture

This is an entity that describes the allowable deviation from perfection (at the micro level) of a surface. The exact definition is given by ANSI and ISO standards.

Vector

A geometric (mathematical) entity that defines a direction in space.

Size tolerance

One of the tolerance entities that applies to the diameter of a hole feature.

Process Instructions

This class of information covers machining steps, feeds & speeds, NC programs, surface treatment instructions and such. This area is quite undefined as structured information today.

Configuration management

This is the information the enterprise requires to control the part. Examples are:

    design control responsibility
    release status
    effectivity
    designer
    checker
    approvals
      .
      .
      .

Some of these items might apply to portions of the part definition (e.g. effectivity of material spec.) as well as to the total part definition.

Flat Plate Model Overview

J S DePace – 17 July 85
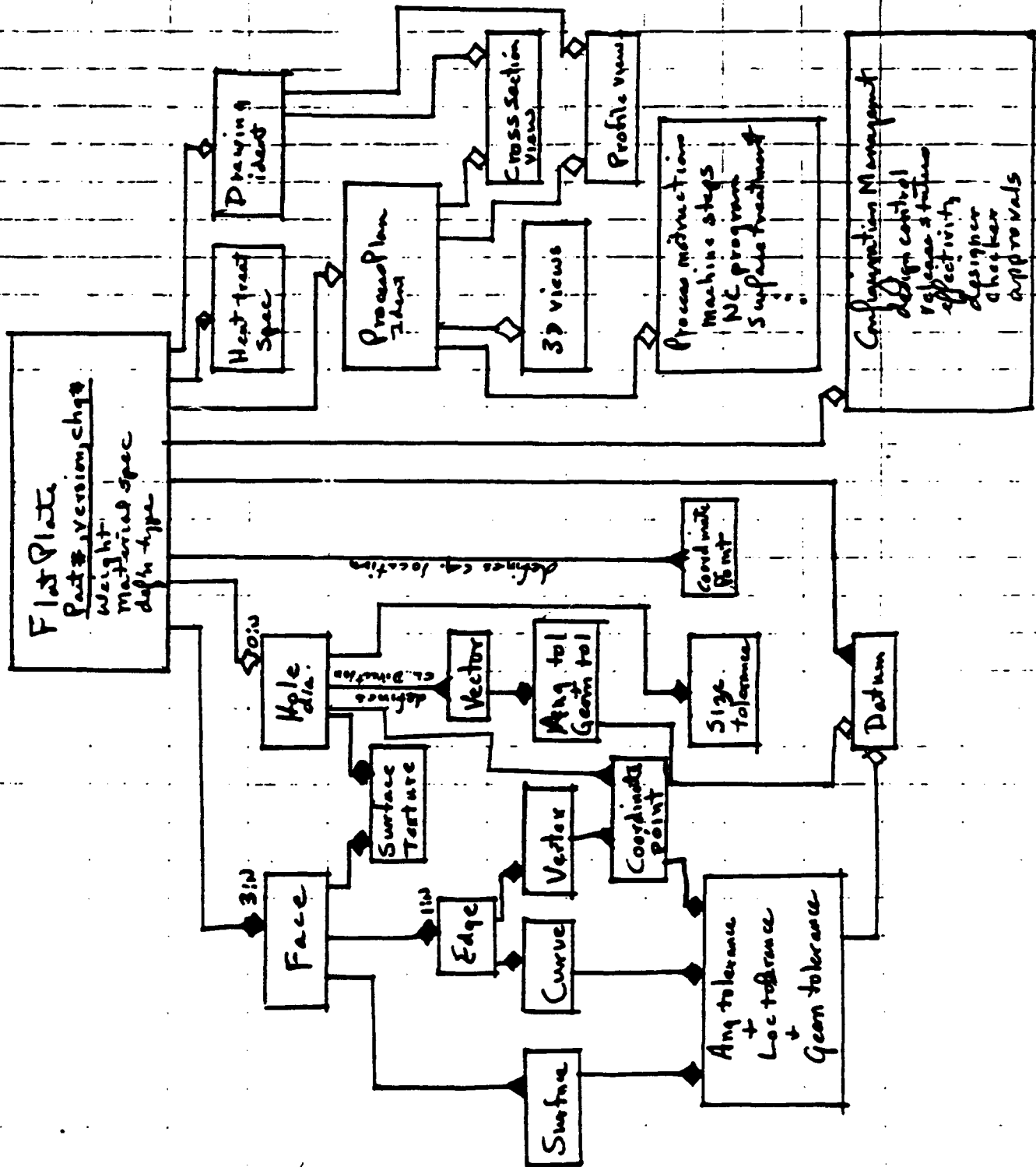
Appendix C3.2

Task 1

Scernarios for Presentation of Flat Plate with Holes

(DISCIPLINE)

Scenarios for Presentation of
Flat Plates with Holes

Submitted by: Don Hemmelgarn, ITI
(513-576-3931)

At the June '85 meeting of the PDES Logical Layer Initiation Task Group it was decided to focus our entity integration efforts on a particular mechanical application to put more 'meat' into the results of that work. The application chosen was the design of flat plates with round holes. It was also agreed that a list of scenarios for viewing flat plate models may be helpful in identifying data requirements (eg. associativities) for the logical layer integration effort. This document represents a first shot at briefly describing various ways flat plate models may need to be presented to the mechanical designer (user view). The intent is to provide information meaningful to the integration of flat plate, wireframe and presentation entities proposed for PDES. However, my feeling is this will more likely serve as impetus for some good discussions concerning PDES entity requirements. For this reason I have included some words about required associativities,etc. with some of the presentation descriptions.

This is by no means an exhaustive list of viewing options for flat plate design. Therefore, I welcome the results of any add/modify/delete operations you may wish to perform on this list.

Each viewing scenario appears as a numbered statement with special data requirements shown in italics.

(1) Need ability to present one to six orthogonal views of the plate, at various scales and positions. This is starting point for description of a flat plate drawing. *Requires view scaling and independent view positioning, in addition to identification of 'model' geometry to be transformed thru the view(s).*

(2) Must be able to display only outside edges(perimeter) of the plate. This is useful for determining material utilization. *Requires association of perimeter attribute to some plate edges.*

(3) Similar to (2), display only cut edges; perimeter, cut-outs, and non-drilled holes for generation of N/C flamecutting data. *Requires association of 'cut' edges.*

(4) Display a cross-section thru the plate model at a given location and orientation. *Requires knowledge of holes or cut-outs and view clipping capability.*

(5) Be able to display or not display geometry representing threads. *Requires entity blanking and association of thread geometry with holes.*

(6) Display surfaces with like finish specifications in the same color. At this point, this means display of color on wireframe plate edge entities. *Requires association of surface finish with plate geometry and ability to assign colors to selected group of entities.*

(7) Display rotated and/or perspective views of the plate geometry.

(8) Hidden lines. Either not display hidden lines or display them as dashed lines. *Requires ability to handle view dependent fonts and inclusion of geometry in one.several, or all defined views.*

(9) Display only drilled holes and their center points for point to point machining operations. *As in (3). requires knowledge of process to generate hole as an attribute for the hole. as well as entity blanking.*

(10) Display or blank a point with an associated text string representing the C.G. of the plate. *Requires a point entity associated with the plate and ability to associate text entity with geometry.*

(11) Display or blank an arrow with associated text string indicating material grain direction.

(12) Display the following textual information along with the views of the plate, at a specified position:
- Part number
- Revision number or code
- Effectivity date
- Material specification

This requires text entity with ability to orient text string and define text characteristics (height. width.etc.). Also requires association of this data with the part being displayed.

Appendix C3.3

Task 1

Flat Plate Qualified Model (NIAM)

(DISCIPLINE)

FLAT PLATE

Part No — Part

Version — Version No

Weight

Flat Plate

Material Spec

Change

Change No

Def'n Type

Size Tol

Face

Hole

D.a.

Surface

defines C.L. direction

Vector

Edge

Surface Texture

Tolerance B

Ang + Geom

Curve

Vertex

Coordinate Point

Flat Plate

Tolerance A

Datum

Ang + Loc + Geom
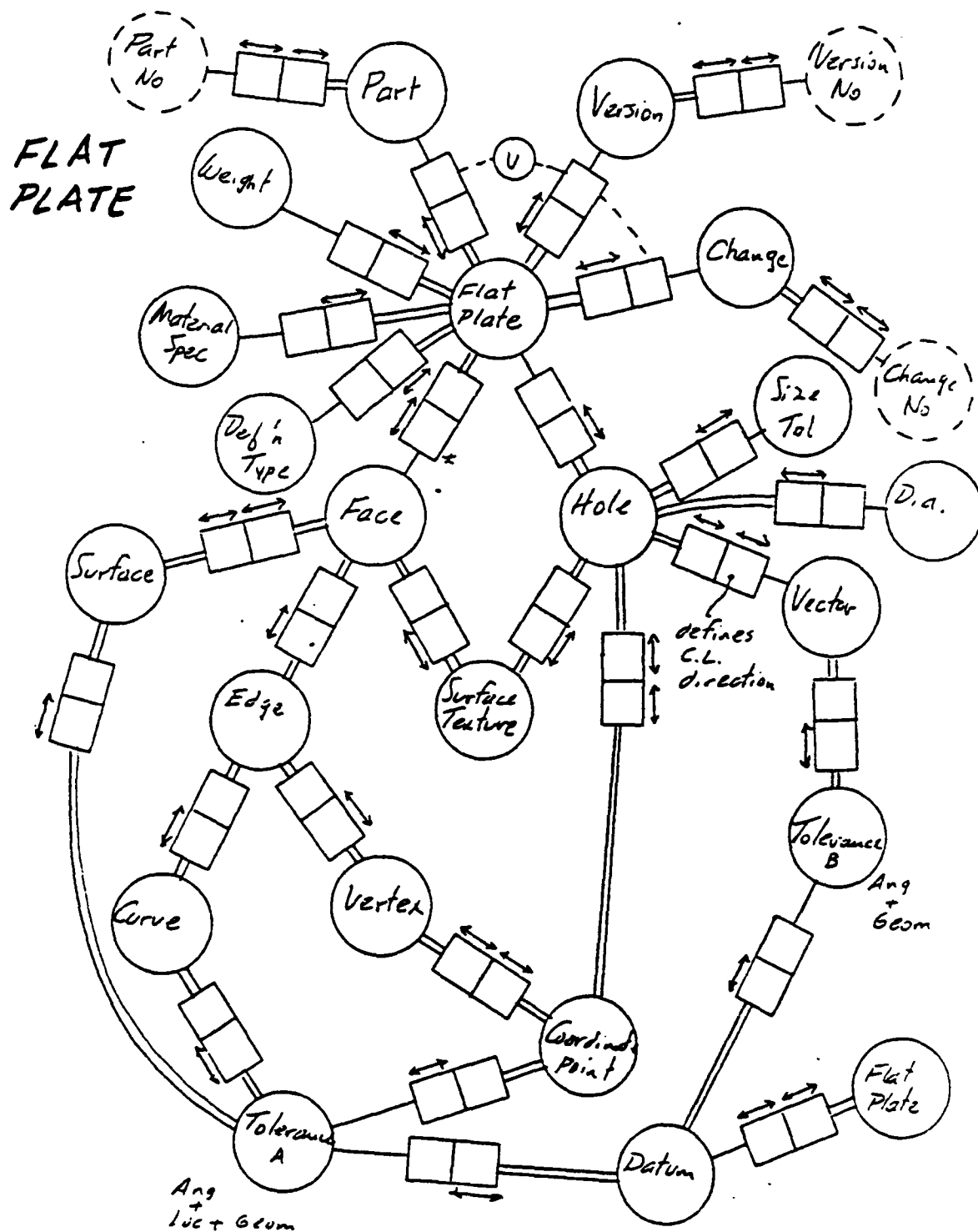
* 3:N cannot be expressed in the MINL subset of NIAM

LITERAL TRANSLATION
AND
QUALIFIED MODEL

Figure 4-10

Apendix C3.4

Task 1

Flat Plate Global Model (NIAM)
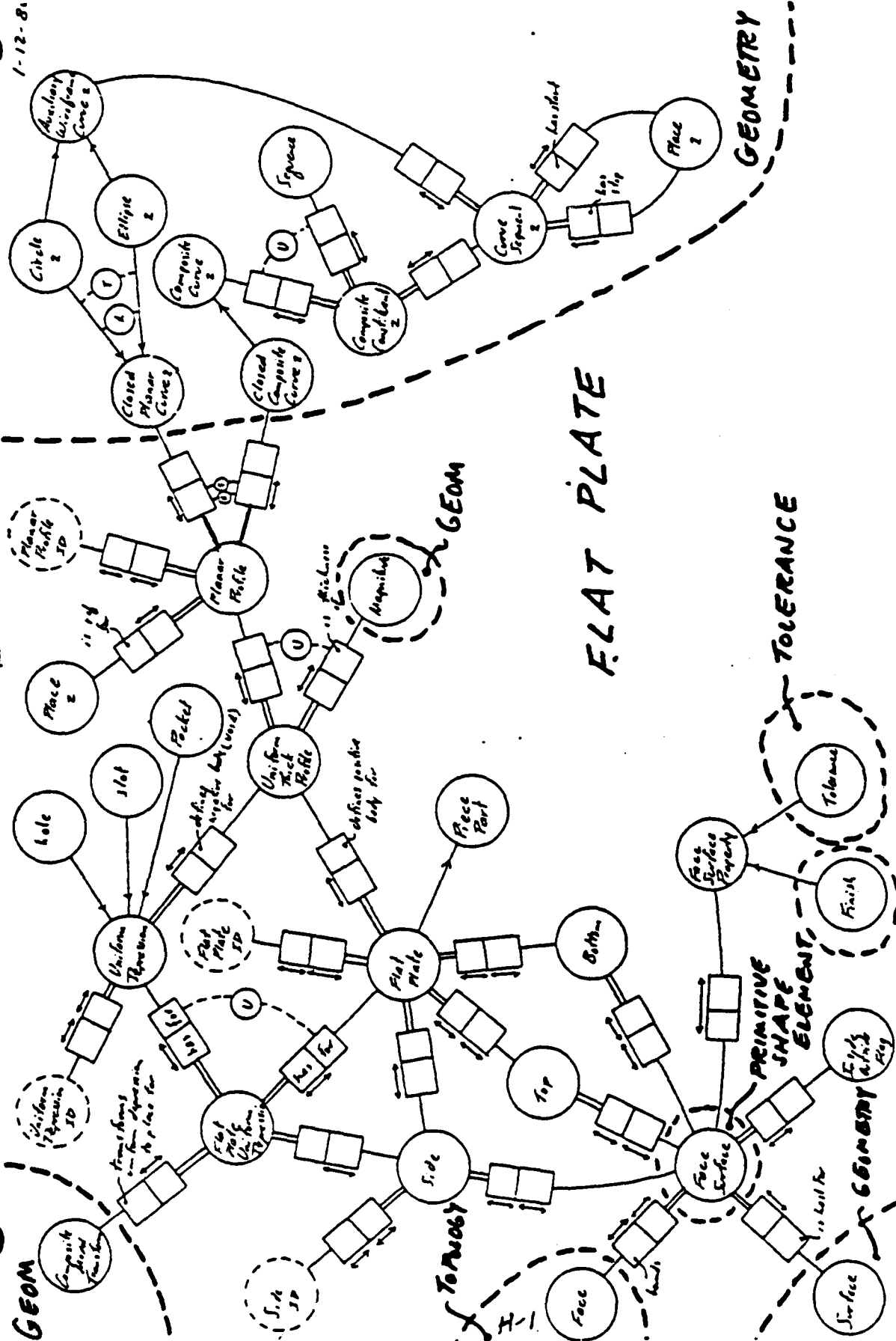
(DISCIPLINE)

GLOBAL MODEL

Figure 4-11

Appendix C3.5

Task 1

<u>Flat Plate Specification Model (DSL)</u>

(DISCIPLINE)

17

```
-- START OF FLAT PLATE ENTITIES
-- this function computes a face based on the input bodies and the
-- parameter value which normally would be 0.0(bottom) or 1.0(top).

FUNCTION calc_uniform_body_face
   (plus:uniform_body;                      -- input/positive body
   minus:LIST(0 TO *) OF uniform_body;   -- input/negative body array
   param:REAL)                              -- input/parametric value
   : face;                                  -- returned/face
BEGIN;
   -- how this is done is left to the reader.
END;


-- this function computes the side face of a uniform-body as
-- specified by side-number.

FUNCTION calc_uniform_body_side
   (plus:uniform_body)                     -- input/positive body
   : LIST(1 TO *) OF face;                 -- returned/face
BEGIN;
   -- how this is done is left to the reader.
END;


-- the uniform-body is a constructive solid based on a sweep of a
-- planar 2d closed curve.

ENTITY uniform_body;
NOROLE
   name                   : OPTIONAL t_name WHERE UNIQUE;
ROLE
   outline                : REFER ( closed_planar_curve2 );
   thickness              : t_magnitude;
END;


ENTITY uniform_flat_plate;
NOROLE
   name                   : OPTIONAL t_name WHERE UNIQUE;
ROLE
   positive_body          : REFER ( uniform_body );
   negative_body          : LIST(0 TO *) OF
                              REFER ( uniform_body );
   top                    : calc_uniform_body_face
                              (positive_body,negative_body,1.0);
   bottom                 : calc_uniform_body_face
                              (positive_body,negative_body,0.0);
   side                   : calc_uniform_body_side_face
                              (positive_body);
END;


END ( * pdes schema * );
```

Appendix C4

Task 2

Electrical Schematic Application Model

(DISCIPLINE)

C4.1   Discipline Model and Documentation

C4.2   Qualified Model (NIAM)

C4.3   Global Model (NIAM)

Appendix C4.1

Task 2

<u>Electrical Schematic Discipline Model and Documentation</u>

(DISCIPLINE)

19

IDEF

STRATEGIC OBJECTIVES

R. W. GALE

SCOPE: This document contains diagrams and text representing a conceptual model of the data required to describe an electrical schematic.

CONTEXT: This model identifies Entities, Relationships and attributes which are logically necessary to construct an electrical schematic. It also identifies some probable relationships to Entities in two other applications: Circuit Analysis and Printed Circuit Board.

PURPOSE: This model proposes portions of the PDES logical layer required to record an electrical schematic for integration with other portions of the PDES logical layer.

VIEWPOINT: The viewpoint of this model is that of the IGES/PDES electrical subcommittee.

PACKAGE CONTENTS

Section	Description
N	Model notes
V	User view: a set of Service Function Requirements (SFR) is used to define the Schematic application and environment.
G	Glossary
FE	Overview: a FEO in unnormalized IDEF-1 form is used to show overall application entity relationships.
ER	Logical Entity Relations: a deviation is made from the IDEF-1 methodology of one entity (plus parents and children) on each sheet to bring the entities into 4 model sub-sections for reader convenience.

INSTRUCTIONS

Please review both User View and the Logical Model to insure that the Schematic and all data requirements have been correctly included. This model is approved as "Recommended" status by the IGES Electrical Subcommittee *Applications Committee.*
If you find a deficiency in the model, please document your suggested corrections and mail to the Committee Chairman:

Larry O'Connell
Sandia National Laboratories
Division 2812
P.O. Box 5800
Albuquerque, NM 87185

VERSION F: Electrical Symbol Instance attributes changed to reflect existance as a Symbol Instance subtype. ER-4 added to redefine "line-ness".

VERSION G: Attribute migration as a result of Version F changes corrected during input to G.D. Metamodeler.

NODE:	TITLE:	NUMBER:
C1		IGES/PDES/G-11-1

Inasmuch as the model intended use is a logical view of the data, independent of existing systems or methods, for development of a neutral product data exchange structure, the reviewer is asked to keep in mind several important modeling constraints.

The entities and attributes are to reflect the _data_ inherent in schematic, as opposed to the _information_ conveyed by a schematic. An example is that the User View preceeding the logical IDEF-1 model describes a netlist. The netlist is here recognized as information assembled by a query against schematic data. In turn, the model must be capable of supporting such a query. Elements of information found in the netlist are found in the Network, Symbol Connection Place and Symbol Instance entity classes.

The model must exist independent of implementations. The entity classes in the logical model must be equally valid for a pencil-drawn schematic or a CAE system with nodal data structures. The Text Template entity found in the IGES PWB model has no validity under this rule and has not been included in this model. The data concept of a "place where a connectior line can terminate" would be valid, and has been included as a Line Connection Place entity.

The model must provide for a union with other related models. This has resulted in entity classes which also belong in other models, but are included to show the key migration and the relation classes. The entities "Product Assembly Definition" and "Component Part" are examples of such entities from other models. .

The entity classes involved in circuit analysis have been developed more than necessary for a schematic because of the high degree of coupling with schematic entities. While packaging requires information from the schematic data, analysis is an interactive data-exchange with the schematic. No requirement for conveying analysis data in a product assembly structure should be assumed by this.

Several entities classes have been created as part of the normalization processes. These entities are not included in the Overview FEO. An example is Electrical Symbol Instance, which provides a home for the Reference Designator (key) attribute. This attribute can be null for some Symbol Instances (e.g., utility symbols), creating a separate entity class.

The author's opinion is also offered that a schematic (all of this model) is not part of a "product definition": Schematics exist as a design aid and as reference documentation.

I.   Definition

The schematic is a symbolic representation of component parts and their electrical connections. The schematic may be for any hierarchical level of product definition, and becomes part of the packaging requirements at that level. Schematics may also contain details of related mechanical nature (e.g., heat sinks, connectors, etc.) and transmission of optical, magnetic or microware energy.

II.   Inputs (sources of design constraints)

o Block diagram, system or subsystem with target block identified

o Interface requirements (e.g., signals and power)   .

o Mechanical package requirements (e.g., chip, hybrid microelectronic assembly or printed board size and mounting)

o Design (and product) constraints and characteristics (e.g., specifications, system equations, test requirements)

o Schedules and/or budget

o Approved (or preferred) parts lists

o Symbol set and drafting standards

III.   Items schematic relates to during design

o Design block diagram

o Boolean operators

o Detail equations/transformations

o Static or dynamic models and simulations

IV.   Schematic constituents

SYMBOLS: a 2-dimensional figure commonly accepted as a representation for a part's functionality.

SYMBOL IDENTIFICATION: part family, part number, part values, part tolerance and reference designator (identifies an instance of the part, and may be later changed to match physical package assignment)

SYMBOL CONNECTION POINTS: indicates where connections to symbol can occur.

SYMBOL CONNECTION POINT IDENTIFICATION: Function code (e.g., $\overline{Q}$, Q, D1, D2, VCC, GND) and pin number (the latter may be different after physical design is completed)

JUNCTION POINT: Symbol (usually a filled small circle) indicating an electrical union of 2 or more connection lines

CONNECTION LINES, SINGLE: a 2-D line or string between symbol connection points or junction points

CONNECTION LINES, BUS: a 2-D line or string (usually bold or highlighted) which represents a collection of connection lines

CONNECTION LINE IDENTIFICATION: signal name (single), bus name or trunk ID (bus)

UTILITY SYMBOLS: non-part symbols for simplification purposes (e.g., off page connector, ground connection, power connection, test point, antenna)

FUNCTION SYMBOL: identification of (usually) non-electrical requirements or parts (e.g., optical energy source, heat sink)

V. Schematic outputs

PICTORIAL: used to review circuit design and as part of final product documentation

NET LIST: a topology derived by examining the logical relations (links) created by the connection lines (joins). Resultant list must be formatted, and contain information, for each use (e.g., physical packaging process, circuit analysis methodology, test development system). Minimal net list information includes part identification (which must match identification of parts in the library of each using system), symbol connection point identification and unique link identification. May be augmented with electrical characteristics (waveform, delay, etc.).

BILL-OF-MATERIALS: a list of parts cited in schematic. Used for stress analysis, packaging the physical circuit and in documentation parts lists.

IDEF

ANALYSIS: The modeling of the electrical functions on a network based on the characteristics of parts connected to that network and the electrical functions occurring on the networks connected to those parts.

ANALYSIS MODEL DATA: The identification of an analysis with the required part model data.

BULLET: A small geometric symbol which may be located at an Intersection Point to indicate that two or more Connection Lines belong to the same Network.

BUS LINE: A piecewise, continuous form of a 2-D line (aka "string") which represents a collection of connection lines. *(STRING written above BUS LINE)*

CIRCUIT CHARACTERISTICS: The analysis-defined activity of a network.

CIRCULAR ARC: A narrow, evenly curved mark drawn or projected.

COMPONENT MODEL: The identification of model data with component parts.

COMPONENT PART: A part which is a constituent part of the defined level.

COMPONENT SCHEMATIC: The hierarchical construct of the schematic of a part represented in a schematic.

CONNECTION GEOMETRY: Geometry which represents the logic of an electrical joining.

ELECTRICAL SYMBOL INSTANCE: A symbol which represents a specific electrical part/function in a schematic.

INTERSECTION PLACE: A place where Connection Lines* are considered to be connected.

LINE: A narrow, elongated mark drawn or projected.

LINES: A piecewise, continuous form of a 2-D line (*aka Polyline). *(STRING written above LINES)*

CONNECTION PLACE: A 2-D location which represents the place where one or more connection Lines* may begin or end.

MODEL DATA: The parameters for a given part required by an analysis package to simulate the function of that part.

MODEL LIBRARY: The reference index for part performance parameters as required for a using circuit analysis process.

NETWORK: A collection of Lines which are defined to be at the same potential (aka Node). *(places written above Lines)*

PRODUCT ASSEMBLY DEFINITION: That collection of informatl necessary to define a complex (hierarchically defined) product for purposes of documentation, analysis, manufactu ing, testing and procuring.

SCHEMATIC: A symbolic representation of a function

SIGNAL: A potential or current (static or impulse) that conveys energy or information. Attributes belong to Physical Model, with some attributes characterized in Circuit Characteristics E.C.

SYMBOL: A 2-D figure commonly accepted to represent a function.

SYMBOL ARC: The occurrence of an arc in a symbol.

SYMBOL CONNECTION PLACE: The location on a symbol where a connection may occur.

SYMBOL INSTANCE: The occurrence of a symbol on a schematic.

SYMBOL LINE: The occurrence of a line in a symbol.

ATTRIBUTES

INFLECTION POINTS: The places where the unit vector of segmented line (see *LINE*) takes on a new value.
STRING

PRODUCT ASSEMBLY DEFINITION

SCHEMATIC

COMPONENT SCHEMATIC

SYMBOL INSTANCE

ATTRIBUTE REFERENCE DESIGNATOR

SYMBOL

COMPONENT PART

MODEL DATA

SIGNAL

NETWORK

CONNECTION LINE GEOMETRY

ANALYSIS

LINE CONNECTION PLACE

SYMBOL CONNECTION PLACE

INTERSECTION PLACE

has
is
has
has
has
has
has
represents has
is for/has
uses/provides
logically links
is carried on
has/produces
joins
represented by
is (line)

Exposition Only

IDEF₁ - E

USED AT:	AUTHOR: Parks, Nelson	DATE:	READER	DATE	CONTEXT:
Innal	PROJECT:	REV: 12 June, 1985			

IGES SCHEMATIC LOGICAL SCHEMA

NOTES: 1 2 3 4 5 6 7 8 9 10

9 Sept, 1985
9 Oct, 1985

WORKING	
DRAFT	
RECOMMENDED	X
PUBLICATION	

PRODUCT ASSY DEFINITION /1

DOCUMENT NO

(SEE PHYSICAL MODEL)

REFERENCED FUNCTIONALLY BY

IS IN

HAS

SCHEMATIC /2

SCHEMATIC NO

DOCUMENT NO

IS IN

HAS

COMPONENT SCHEMATIC /3

SCH NO .NEXT ASSY (FK)
SCHEMATIC NO (FK)

COMPONENT PART /8

PART NO . DOC NO (FK)
DOCUMENT NO (FK)
(SEE PHYSICAL MODEL)

SYMBOL LIBRARY /4

LIBRARY NAME/ VERS

PART NUMBER - SYMBOL
INDEX

SYMBOL INSTANCE /6

SYM-LOCATION
SHEET NO
SYMBOL-TYPE (FK)
SCHEMATIC NO (FK)
LIBRARY NAME/ VERS (FK)

REPRESENTS

IS A

ELECTRICAL SYM INST /7

SYM-LOCATION (AKI)
REFERENCE DESIGNATOR (AKI)
SHEET NO
SYMBOL-TYPE (FK)
SCHEMATIC NO (FK)
LIBRARY NAME/VERS (FK)

IDENTIFICATION VALUE

CIRCULAR ARC /12

RADIUS
START ANGLE
STOP ANGLE

IS

SYMBOL ARC /11

LOCATION
RADIUS (FK)
START ANGLE (FK)
STOP ANGLE (FK)
SYMBOL TYPE (FK)
LIBRARY NAME/VERS (FK)

LINE WIDTH

SYMBOL /3

SYMBOL-TYPE
LIBRARY NAME/ VERS (FK)

PART FUNCTIONAL TYPE
OR UTILITY FUNCTION

BELONGS

HAS

SYMBOL LINE /9

DIRECTION
LOCATION
SYMBOL-TYPE (FK)
LIBRARY NAME/VERS (FK)
LENGTH (FK)

LINE WIDTH

LINE /10

LENGTH

TITLE: SCHEMATIC, SYMBOL, PART RELATIONS

NUMBER: A/ESCH/6-11 -B

NODE:

IDEF₁ - E...

NETWORK /21

(SEE ER-3)

LOGICALLY LINKS

REPRESENTED BY

SYMBOL INSTANCE /6

(SEE ER-1)

HAS

P

CONNECTION GEOMETRY /15

START/ END PLACE
NETNAME (FK)
SCHEMATIC NO (FK)

LINE WEIGHT

JOINS

P

CONNECTION PLACE /16

LOCATION
CONNECTION TYPE
SCHEMATIC NO (FK)

NETNAME (FK)
START PLACE (FK)
END PLACE (FK)

CONNECTION TYPE

P1

SYMBOL CONN PLACE /18

CP-LOCATION
CONNECTION TYPE
SCHEMATIC NO (FK)
SYMBOL TYPE (FK)
SHEET NO (FK)
LIB NAME/VERS (FK)
FUNCTION CODE
PIN NO (O)
INPUT OR OUTPUT
NETNAME (FK)
SYM-LOCATION (FK)

INTERSECTION PLACE /17

CP-LOCATION
CONNECTION TYPE
SCHEMATIC NO (FK)

HAS

Z

BULLET /19

CP-LOCATION (FK)
SCHEMATIC NO (FK)

SHAPE-SIZE

TITLE: SCHEMATIC CONNECTION RELATIONS

NUMBER: A/ESCH/G-...-9

NODE:

IDEF$_1$ - E$_{mm}$

USED AT:	AUTHOR: Parks, Nelson	DATE:	READER	DATE	CONTEXT:
	PROJECT:	REV: 12 June, 1985			
		9 Sept. 1985			

WORKING	DRAFT	
DRAFT	X RECOMMENDED	
	PUBLICATION	

IGES SCHEMATIC LOGICAL SCHEMA

NOTES: 1 2 3 4 5 6 7 8 9 10

COMPONENT PART /8
PART NO
DOCUMENT NO (FK)

SYMBOL TYPE
OTHERS (SEE PHYSICAL MODEL)

MAB

COMPONENT MODEL /27
PART NO (FK)
MOD LIB NAME (FK)
MODEL ID (FK)

MODEL LIBRARY /23
MOD LIB NAME

PART NO-PARAM INDEX

CATALOGS

MODEL DATA /25
MODEL ID
MOD LIB NAME (FK)

STIMULUS VALUE
UNITS
TRANSFER FUNCTION

IS

PROVIDES

ANALYSIS MODEL DATA /26
ANAL NAME/VERS (FK)
MODEL ID (FK)
MOD LIB NAME (FK)

USES

ANALYSIS /24
ANALYSIS NAME/VERS

INDEX
ALGORITHM

USED

PRODUCES

SCHEMATIC/
(SEE ER-1)

MAB

NETWORK /21
NETNAME
SCHEMATIC NO (FK)

SIGNAL (FK)
ANALYSIS NAME/VERS (FK)

MAB

CIRCUIT CHARACTERISTICS /22
SEQUENCE NO
NETNAME (FK)
ANAL-NAME/VERS (FK)
SCHEMATIC NO (FK)

f(|v(i)|), g(i(i))

SIGNAL
(SEE ELECTRICAL PRODUCT MODEL)

IS CARRIED ON

REPRESENTED BY

CONNECTION GEOMETRY /15
(SEE ER-2)

19.103.1983 SCH-108P

ER-3

NUMBER: A/ESCH/F = 11 -10

TITLE: NETWORK ANALYSIS RELATIONS.

NODE:

IDEF₁ - E

LINE /10

SEE ER-1

CONNECTION GEOMETRY /15

SEE ER-2

BUS STRING /13

SCHEMATIC NO (FK)
LENGTH (FK)
START/END PLACE (FK)
NETNAME (FK)

B-INFLECTIONS
WIDTH
BUS NAME

STRING /14

SCHEMATIC NO (FK)
LENGTH (FK)
START/END PLACE (FK)
NETNAME (FK)

S-INFLECTIONS

NODE:

TITLE: RELATIONS

NUMBER:

In-4

Appendix C4.2

Task 2

<u>Qualified Electrical Schematic Model (NIAM)</u>

(DISCIPLINE)
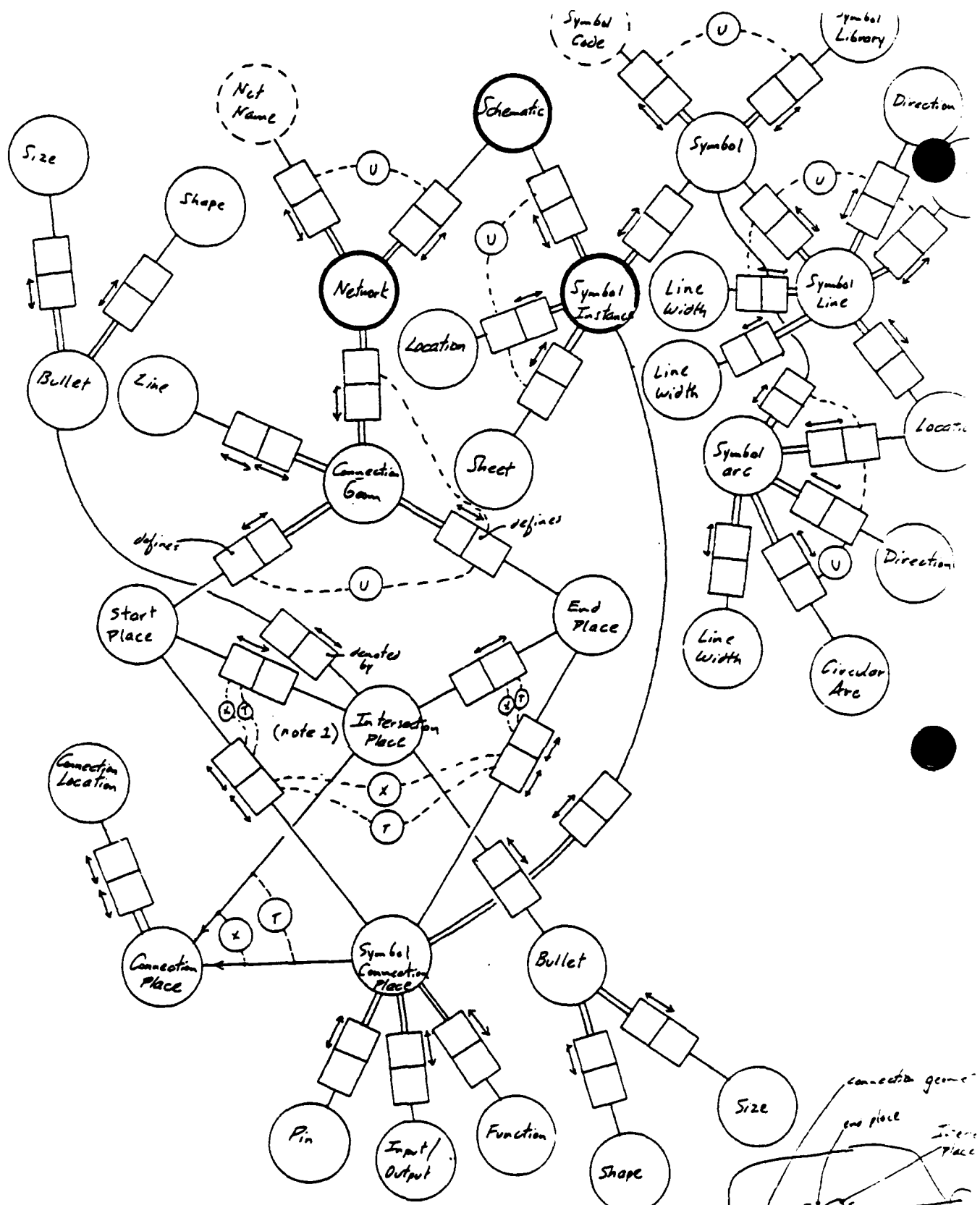
note 1: All Start and End places associated with Intersection Places are considered to be at the same place. They may in actuality be at slightly different places.

ELECTRICAL
QUALIFIED

Appendix C4.3

Task 2

Electrical Schematic Global Model (NIAM)

(DISCIPLINE)

Presentation

GEOMETRY

Presentation

Topology
defines ideal connectivity
allows definition of connectivity without defining geometry — ec
design place of for commun... topology.

Geometry

GTA

Note 1. All Place 2's associated with Intersection are considered to be at the same place.

ELECTRICAL
GLOBAL

Presentation

Appendix C5

Task 2

<u>Tolerancing Discipline Model</u>

(DISCIPLINE)

C5.1   Discipline Model and Documentation

C5.2   Qualified Model (NIAM)

C5.3   Global Model (NIAM)

C5.4   Specification Model (NIAM)

Appendix C5.1

Task 2

Tolerancing Discipline Model and Documentation

(DISCIPLINE)

Scope

This document provides the definition of functional content for tolerancing practices as specified by ANSI Y14.5M-1982 and ISO 1101 and 1660. These specifications are considered to be functionally identical for the purposes of this effort.

This information model is intended to completely define all tolerance information specified by ANSI Y14.5M-1982 and the corresponding ISO specifications. Any deficiencies should be commented on in writing.

Excluded from the scope

Computing tolerances

Process tolerances

Pictorial representations of tolerances

Dimensioning practices.

Non-mechanical part tolerances (e.g.electrical component values)

Assumptions

Product models are assumed to be 3D wireframes with surfaces.

Models contain exact definitions of nominal product geometry.

Dimension information is implicit in model geometry.

Topology constructs, where used in this model, are included to satisfy the functional requirements of tolerancing. Since a completely surfaced wireframe and a BREP model are essentially equivalent, we have adopted the BREP terminology of face, edge and vertex in our tolerance model.

Many entity classes defined here are known to be incomplete. Usually these classes are dependent on other reference models that are not available yet.

General Definitions

Product

An item(s) which is(are) manufactured, or used in the manufacture of an another item.

Model

A digital definition of a product.

Drawing

A pictorial representation of a product.

Dimension

A numerical value which is a measure of the product implicit, in the model geometry.

Notation

Where entities are referenced that are not properly tolerance entities (e.g. conic arc), those entity references are marked with an "*". No definitions are provided for those entities.

The Pascal-like "data structure" notation used herein is not intended in any way to indicate or specify any particular physical representation of the information. These structures are included only to clarify the intent and content of the information model.

(100) Geometry

Geometry elements or entities are mathematically exact constructs such as points, lines and the like, that represent topological elements which are physically present on a part. Geometry defined here is only that explicitly contained in the tolerance information model.

```
                Class of: Unit_Vector (101)
                          Point_Vector (102)
                          Vector (103)
                          Curve (106)
                          Coordinate (107)
                          Point *
                End_class;
```

(101) Unit vector

A sequence of three real values that are the direction cosines of a unit vector. The Euclidean norm is exactly 1.

```
                Start_entity
                i_Cosine            : Real
                j_Cosine            : Real
                k_Cosine            : Real
                End_entity;
```

(102) Point vector

A point vector is three real values that are direction cosines, a coordinate point through which the vector passes and a length value.

```
                Start_entity
                Position            : Coordinate (107)
                Direction           : Vector (103)
                End_entity;
```

(103) Vector

A vector is three real values that are direction cosines and a length value.

```
                Start_entity
                Direction           : Unit_Vector (101)
                Length              : Real
                End_entity;
```

(106) Curve

A curve is a class of entities. These entities are assumed to be defined by the information model for 3D surfaced wire frames.

```
                    Class of: line *
                              circular arc *
                              conic arc *
                    End_class;
```

(107) Coordinate

A set of three real values that specifies a theoretical position in space.

```
                    Start_entity
                    X_coordinate        : Real
                    Y_coordinate        : Real
                    Z_coordinate        : Real
                    End_entity;
```

(200) Topology

Topology elements or entities are constructs that define touchable or viewable portions of a part. Topology elements are defined in terms of geometry elements and other topology elements.

```
          Class of: vertex (201)
                    edge (202)
                    face (203)
          End_class;
```

(201) Vertex

A vertex defines logical connectivity of edges.

```
          Start_entity
          Point_reference      : Point *
          End_entity;
```

(202) Edge

An edge bounds a curve and defines logical connectivity of faces.

```
          Start_entity
          Curve_reference      : Curve (106)
          Start_point          : Vertex (201)
          End_point            : Vertex (201)
          End_entity;
```

A Circular_edge and Planar_edge are qualified types of edge.

(203) Face

A face is a bounded geometric surface.

```
          Start_entity
          Surface_reference    : Surface *
          Boundary             : Array (1 to maxint) of
                                   Edge (202)
          End_entity;
```

Circular_face, Planar_face, Cylindrical_face, Disk_face and Runout_face are qualified types of face.

(204) Circular edge

Circular edge is an obvious subset of Edge (202).

(205) Circular face

A circular face is a subset of Face (203). A circular face exhibits a circular cross section in any plane perpendicular to the axis of symmetry.

(206) Planar face

A planar face is a subset of Face (203). A planar face references a surface that is planar in the region of interest.

(207) Cylindrical face

A cylindrical face is a subset of Face (203). The surface referenced by the face must be cylindrical in the region of interest.

(208) Runout face

A runout face is a subset of Face (203). The subset consists of circular faces (205) and disk faces (211).

(209) Planar edge

A subset of Edge (202). Planar edges have all constituent elements in a single plane.

(211) Disk face

A subset Face (203). A disk face refers to a surface perpendicular to an axis of symmetry with a circular boundary (edge).

(300) Feature

A feature is a referencable geometric subset of interest in a model. A feature is also a class of entities:

```
        Class of: face (203)
                  edge (202)
                  vertex (201)
                  datum (301)
                  size_feature (303)
                  form_feature (304)
        End_class;
```

(301) Datum

A theoretically exact geometric reference to which toleranced features are related.

```
        Start_entity
        Referenced_entity   : Topology (200) or Feature_of
                                _size (306)
        Name                : String(2)
        End_entity;
```

Attribute Descriptions:

   Referenced_entity: An entity which serves as the
        exact definition of the datum.

   Name: An alphabetic string that provides a unique
        designation for the datum (range: A..Z AA..ZZ,
        excluding I, O, and Q)

(302) Conditioned Datum

A conditioned datum is a datum to which a material condition specification applies.

```
        Start_entity
        Base                : Datum (301)
        Material_condition  : (M, L, S)
        End_entity;
```

Attribute Descriptions:

   Base: A datum entity which serves as the exact
        definition of the datum.

   Material_condition: An enumerated list that indicates
        the material condition at which the tolerance
        applies: M - maximum material condition; L -
        least material condition; or S - regardless of
        feature size.

least material condition; or S - regardless of feature size.

(303) Size feature

A feature whose tolerancable geometric location is derived from physical feature geometry and is symmetric about a point, axis, curve or surface. Size tolerances of the feature are independent of feature location. This entity exists to allow the application of tolerances to partially modeled or incomplete Features_of_size.

```
Start_entity
Tolerance_entity      : Array (2 to maxint) of
                           face (203)
End_entity;
```

(304) Form feature

A referencable geometric subset whose name implies a specific physical configuration. The subset does not include the size feature entities.

```
Class of: Hole (305)
          Feature_of_size (306)
End_class;
```

(305) Hole

A circular opening, of possibly stepped or graduated diameter, into or through a workpiece. It is oriented at an angle, possibly normal, to the surface of the workpiece.

(306) Feature of size

A feature of size is a subset of Form feature (304). An example is the Hole (305).

(400) Tolerance

The allowable deviation of a geometric aspect of a product from its
design nominal geometry.

                    Class of: coordinate_tolerance (401)
                              geometric_tolerance (402)
                    End_class;

(401) Coordinate tolerance

A coordinate tolerance is a class of entities:

                    Class of: angle_tolerance (404)
                              location_tolerance (403)
                              size_tolerance (405)
                    End_class;

(402) Geometric tolerance

A geometric tolerance is a class of entities:

                    Class of: angularity (406)
                              circular_runout (407)
                              circularity (408)
                              concentricity (409)
                              cylindricity (410)
                              flatness (411)
                              parallelism (412)
                              perpendicularity (413)
                              position (414)
                              profile_of_a_line (415)
                              profile_of_a_surface (416)
                              straightness (417)
                              total_runout (418)
                    End_class;

(403) Location tolerance

Allowable deviation of measure of a feature from its design nominal
position relative to a specified base location along a specified path.

```
          Start_entity
          Origin                : Location_origin (503)
          Path                  : Location_path (504)
          Toleranced_entity     : Array (1 to maxint) of
                                  Location_toleranced_entity (502)
          Basic                 : Boolean
          Plus_Tol              : Real
          Minus_Tol             : Real
          End_entity;
```

Attribute Descriptions:

Origin: An entity that serves as the base or origin
    of the calculated dimension. It is the "from"
    entity of the directed dimension.

Path: A curve along which (or in the direction of
    which) the dimension is measured.

Toleranced_Entity: An array of entities to which
    the tolerance applies.

Basic: A boolean (true/false) flag that indicates
    whether the entity is used as a BASIC dimen-
    sion.

Plus_Tol: The absolute value of the tolerance that is
    added to the nominal dimension value to esta-
    blish the maximum allowable deviation of the
    toleranced entity from the nominal.

Minus_Tol: The absolute value of the tolerance that
    is subtracted from the nominal dimension value
    to establish the minimum allowable deviation of
    the toleranced entity from the nominal.

(404) Angle tolerance

Allowable deviation of measure of a toleranced entity from its design
nominal orientation relative to a specified base orientation.

```
                    Start_entity
                    Origin              : Angle_origin (501)
                    Toleranced_entity   : Array (1 to maxint) of
                                          Angle_toleranced_entity (500)
                    Basic               : Boolean
                    Plus_Tol            : Real
                    Minus_Tol           : Real
                    Sense               : Boolean
                    End_entity;
```

Attribute Descriptions:

Origin: An entity that serves as the base or origin
of the calculated dimension. It is the "from"
entity of the directed dimension.

Toleranced_Entity: An array of entities to which the
tolerance applies.

Basic: A boolean (true/false) flag that indicates
whether the entity is used as a BASIC dimen-
sion.

Plus_Tol: The absolute value of the tolerance that is
added to the nominal dimension value to esta-
blish the maximum allowable deviation of the
toleranced entity from the nominal.

Minus_Tol: The absolute value of the tolerance that
is subtracted from the nominal dimension value
to establish the minimum allowable deviation of
the toleranced entity from the nominal.

Sense: A boolean (true/false) flag which indicates
whether or not the angle is measured in a
counter-clockwise manner. The orientation of
measure is determined by the cross-product from
the origin to the toleranced entity vector.
If the cross-product is zero, the path vector
is required to determine the orientation.

(405) Size tolerance

Allowable deviation of measure of a toleranced entity from its design nominal magnitude, e.g. hole diameter.

```
            Start_entity
            Toleranced_entity    : Array (1 to maxint) of
                                     Size_toleranced_entity (505)
            Basic                : Boolean
            Plus_Tol             : Real
            Minus_Tol            : Real
            End_entity;
```

Attribute Descriptions:

Toleranced_Entity: An array of entities to which the tolerance applies.

Basic: A boolean (true/false) flag that indicates whether the entity is used as a BASIC dimension.

Plus_Tol: The absolute value of the tolerance that is added to the nominal dimension value to establish the maximum allowable deviation of the toleranced entity from the nominal.

Minus_Tol: The absolute value of the tolerance that is subtracted from the nominal dimension value to establish the minimum allowable deviation of the toleranced entity from the nominal.

(406) Angularity

Angularity is the condition of a surface or axis at a specified angle
(other than 90 degrees) from a datum plane or axis. An Angularity
tolerance specifies a tolerance zone defined by two parallel planes at
the specified basic angle from the datum plane or axis within which
the surface of or axis of the considered feature must lie.
See ANSI Y14.5M 1982, page 106, section 6.6.2
See ISO 1101, pages 18-19, section 5.9

```
                    Start_entity
                    Toleranced_entity   : Array (1 to maxint) of
                                      Angularity_toleranced_entity (506)
                    Tolerance           : Real
                    Material_condition  : (M, L, S)
                    Projection          : Optional Point_vector (102)
                    Primary_datum       : Conditioned_datum (302)
                    Secondary_datum     : Optional Conditioned_datum (302)
                    Tertiary_datum      : Optional Conditioned_datum (302)
                    End_entity;
```

Attribute Descriptions:

Toleranced_Entity: An array of entities to which
        the tolerance applies.

Tolerance: The specified allowable deviation from
        the design nominal value.

Material_condition: An enumerated list that indicates
        the material condition at which the tolerance
        applies: M - maximum material condition; L -
        least material condition; or S - regardless of
        feature size.

Projection: A point_vector which specifies the
        additional height and direction of the
        projected tolerance zone outside the feature
        boundary.
        Note: Must be parallel to the toleranced
              features. Length of vector determines the
              extent of the zone.

Primary_datum: A conditioned_datum entity from which
        the dimension is measured. This primary datum
        is the most important datum relative to the
        tolerance.

Secondary_datum: A conditioned_datum entity which is
        the second most important datum relative to the
        tolerance.

Tertiary_datum: A conditioned_datum entity which is the
        third most important datum relative to the
        tolerance. With the primary and secondary data,

it establishes a datum reference frame that exactly
locates the toleranced feature.

(407) Circular runout

Circular runout tolerance defines the maximum allowable deviation of position of a coordinate on the toleranced feature during one complete revolution of that feature about the datum axis, without relative axial displacement of the measuring position.   Where applied to surfaces constructed around a datum axis, it is used to control the cumulative variations of circularity amd coaxiality. Where applied to surfaces constructed at right angles to the datum axis, circular runout controls circular elements of a plane surface.
See ANSI Y14.5M 1982, page 109, section 6.7.2.1
See ISO 1101, page 22-23, section 5.12

```
                    Start_entity
                    Toleranced_entity    : Array (1 to maxint) of
                              Circular_runout_toleranced_entity (507)
                    Tolerance            : Real
                    Primary_datum        : Datum (301)
                    Co-primary_datum     : Optional Datum (301)
                    End_entity;
```

Attribute Descriptions:

Toleranced_Entity: An array of entities to which
     the tolerance applies.

Tolerance: The specified allowable deviation from
     the design nominal value.

Primary_datum: A datum entity from which
     the dimension is measured. This primary datum
     is the most important datum relative to the
     tolerance.

Co-primary_datum: An additional datum entity which
     establishes an axis with the primary datum.

(408) Circularity

Circularity is a condition of a surface of revolution such that all
points of the surface intersected by a plane perpendicular to the axis
or center point are equidistant from the intersection point of the
plane and the axis or center point. A Circularity tolerance defines
the distance between two concentric circles within which the
coordinates of the toleranced feature must lie. These circles are
perpendicular to and centered on the axis of the toleranced feature.
See ANSI Y14.5M 1982, page 95, section 6.4.3
See ISO 1101, page 14, section 5.3

```
                    Start_entity
                    Toleranced_entity   : Array (1 to maxint) of
                                        Circularity_toleranced_entity (508)
                    Tolerance           : Real
                    End_entity;
```

        Attribute Descriptions:

            Toleranced_Entity: An array of entities to which
                the tolerance applies.

            Tolerance: The specified allowable deviation from
                the design nominal value.

(409) Concentricity

A Concentricity tolerance specifies the diameter of a cylinder cen-
tered on a datum point or axis within which the center or axis of the
toleranced circular or cylindrical feature must lie. The specified
tolerance and datum reference apply only on a Regardless-of-Feature-
Size basis.
See ANSI Y14.5M 1982, page 84, section 5.11.3
See ISO 1101, page 21, section 5.11.1

```
                    Start_entity
                    Toleranced_entity   : Array (1 to maxint) of
                                Concentricity_toleranced_entity (509)
                    Tolerance           : Real

                    Primary_datum       : Datum (301)
                    Co-primary_datum    : Optional Datum (301)
                    End_entity;
```

        Attribute Descriptions:

            Toleranced_Entity: An array of entities to which
                the tolerance applies.

            Tolerance: The specified allowable deviation from
                the design nominal value.

            Cylindrical_zone: A boolean (true/false) flag that
                indicates that the tolerance value is the dia-
                meter of a cylindrical zone within which the
                axis or line must lie. If false, the zone is
                parallelepipedic or the space between two
                parallel lines or planes.

            Primary_datum: A datum entity from which
                the dimension is measured. This primary datum
                is the most important datum relative to the
                tolerance.

            Co-primary_datum: An additional datum entity which
                establishes an axis with the primary datum.

(410) Cylindricity

Cylindricity tolerance defines the distance between two coaxial cylinders within which the toleranced cylindrical feature must lie. Unlike circularity, the tolerance applies simultaneously to both circular and longitudinal elements of the surface.
See ANSI Y14.5M 1982, page 96, section 6.4.4
See ISO 1101, page 14, section 5.4

```
            Start_entity
            Toleranced_entity   : Array (1 to maxint) of
                        Cylindricity_toleranced_entity (510)
            Tolerance           : Real
            End_entity;
```

Attribute Descriptions:

Toleranced_Entity: An array of entities to which
        the tolerance applies.

Tolerance: The specified allowable deviation from
        the design nominal value.

(411) Flatness

Flatness tolerance defines distance between two parallel planes
between which the toleranced surface must lie. The tolerance may be
applied on a unit basis as a means of preventing abrupt surface
variation within a small area of the feature.
See ANSI Y14.5M 1982, page 94, section 6.4.2
See ISO 1101, page 13, section 5.2

                    Start_entity
                    Toleranced_entity   : Array (1 to maxint) of
                                           Flatness_toleranced_entity (511)
                    Tolerance   : Real
                    Material_condition   : (M, L, S)
                    Per_unit_square   : Optional Real
                    End_entity;

          Attribute Descriptions:

               Toleranced_Entity: An   array   of entities to which
                    the tolerance applies.

               Tolerance: The   specified allowable deviation   from
                    the design nominal value.

               Material_condition: An enumerated list that indicates
                    the material condition at which the tolerance
                    applies: M - maximum material condition; L -
                    least material condition; or S - regardless of
                    feature size.

               Per_unit_square: Specifies the side of any square
                    region   on   the   toleranced_entity   over which
                    the tolerance value applies. Used to prevent
                    abrupt surface variation in a relatively small
                    area of the feature.

(412) Parallelism

Parallelism is the condition of a surface equidistant at all points
from a datum plane or an axis equidistant along its length from a
datum axis. A Parallelism tolerance specifies the distance between two
planes or lines parallel to a datum plane, or axis, within which the
line elements of the surface or axis of the considered feature must
lie, or a cylindrical tolerance zone whose axis is parallel to the
datum axis, within which the axis of the considered feature must lie.
The allowable feature position zone may be cylindrical(fig 45), paral-
lelepipedic (fig 51) or planar (fig 47,54) for line features and is
similar to a flatness tolerance zone for surface features (fig 57,60).
These figure references are to the ISO 1101 standard.
See ANSI Y14.5M 1982, page 106, section 6.6.3
See ISO 1101, page 15-17, section 5.7

```
                    Start_entity
                    Toleranced_entity   : Array (1 to maxint) of
                                          Parallelism_toleranced_entity (512)
                    Tolerance           : Real
                    Material_condition  : (M, L, S)
                    Cylindrical_zone    : Boolean
                    Projection          : Optional Point_vector (102)
                    Primary_datum       : Conditioned_datum (302)
                    End_entity;
```

Attribute Descriptions:

   Toleranced_Entity: An array of entities to which
       the tolerance applies.

   Tolerance: The specified allowable deviation from
       the design nominal value.

   Material_condition: An enumerated list that indicates
       the material condition at which the tolerance
       applies: M - maximum material condition; L -
       least material condition; or S - regardless of
       feature size.

   Cylindrical_zone: A boolean (true/false) flag that
       indicates that the tolerance value is the dia-
       meter of a cylindrical zone within which the
       axis or line must lie. If false, the zone is
       parallelepipedic or the space between two
       parallel lines or planes.

   Projection: A point_vector which specifies the
       additional height and direction of   the
       projected tolerance zone outside the feature
       boundary.
       Note: Must be parallel to the toleranced
             features. Length of vector determines the
             extent of the zone.

- 20 -

Primary_datum: A conditioned_datum entity from which
    the dimension is measured.

(413) Perpendicularity

Perpendicularity tolerance defines the allowable linear deviation from
a true right angle of line or surface features with respect to line or
surface datums. Several interpretations of the exact tolerance zone
are possible for line feature with respect to surface datums. See
figures 65,67 and 69 in ISO 1101,page 17.
See ANSI Y14.5M 1982, page 105, section 6.6.4
See ISO 1101, page 17-18, section 5.8

```
            Start_entity
            Toleranced_entity   : Array (1 to maxint) of
                        Perpendicularity_toleranced_entity (513)
            Tolerance           : Real
            Material_condition  : (M, L, S)
            Cylindrical_zone    : Boolean
            Projection          : Optional Point_vector (102)
            Primary_datum       : Conditioned_datum (302)
            Secondary_datum     : Optional Conditioned_datum (302)
            End_entity;
```

Attribute Descriptions:

.   Toleranced_Entity: An array of entities to which
        the tolerance applies.

Tolerance: The specified allowable deviation from
    the design nominal value.

Material_condition: An enumerated list that indicates
    the material condition at which the tolerance
    applies: M - maximum material condition; L -
    least material condition; or S - regardless of
    feature size.

Cylindrical_zone: A boolean (true/false) flag that
    indicates that the tolerance value is the dia-
    meter of a cylindrical zone within which the
    axis or line must lie. If false, the zone is
    parallelepipedic or the space between two
    parallel lines or planes.

Projection: A point_vector which specifies the
    additional height and direction of the
    projected tolerance zone outside the feature
    boundary.
    Note: Must be parallel to the toleranced
        features. Length of vector determines the
        extent of the zone.

Primary_datum: A conditioned_datum entity from which
the dimension is measured. This primary datum
is the most important datum relative to the
tolerance.

Secondary_datum: A conditioned_datum entity which is
the second most important datum relative to the
tolerance.

(414) Position

Position tolerance defines the allowable deviation of position of the
center point, axis, or center plane of a feature of size. A center
point tolerance defines the diameter of a spherical or circular zone,
an axis tolerance defines the measure of a cylindrical, parallelepi-
pedic or planar zone and a center plane tolerance defines a zone
specified by the distance between two bounding parallel planes. Each
zone is considered to contain the true position of the toleranced
feature.
See ANSI Y14.5M 1982, page 53-89, section 5.2
See ISO 1101, page 19-20, section 5.10

```
                    Start_entity
                    Toleranced_entity    : Array (1 to maxint) of
                                           Position_toleranced_entity (514)
                    Tolerance            : Real
                    Material_condition   : (M, L, S)        .
                    Cylindrical_zone     : Boolean
                    Projection           : Optional Point_vector (102)
                    Primary_datum        : Conditioned_datum (302)
                    Secondary_datum      : Optional Conditioned_datum (302)
                    Tertiary_datum       : Optional Conditioned_datum (302)
                    End_entity;
```

Attribute Descriptions:

Toleranced_Entity: An array of entities to which
the tolerance applies.

Tolerance: The specified allowable deviation from
the design nominal value.

Material_condition: An enumerated list that indicates
the material condition at which the tolerance
applies: M - maximum material condition; L -
least material condition; or S - regardless of
feature size.

Cylindrical_zone: A boolean (true/false) flag that
indicates that the tolerance value is the dia-
meter of a cylindrical zone within which the
axis or line must lie. If false, the zone is
parallelepipedic or the space between two
parallel lines or planes.

Projection: A point_vector which specifies the
        adoitional   height   and direction   of   the
        projected tolerance zone outside the feature
        boundary. Note: The length of the vector
        determines the length of the zone.

Primary_datum: A conditioned_datum entity from which
        the dimension is measured. This primary datum
        is the most important datum relative to the
        tolerance.

Secondary_datum: A conditioned_datum entity which is
        the second most important datum relative to the
        tolerance.

Tertiary_datum: A conditioned_datum entity which is the
        third most important datum relative to the
        tolerance.   With the primary and secondary data,
        it establishes a datum reference frame that exactly
        locates the toleranced feature.

(415) Profile of a line

Profile of a line tolerance specifies the diameter of a circle which,
when its center, or one tangent, moves along the design nominal curve
feature, sweeps the region in which the feature must lie. The
tolerance is two-dimensional and applies normal (perpendicular) to the
true profile at all points. Where a sharp corner is included, the
tolerance zone extends to the intersection of the boundary lines.
See ANSI Y14.5M 1982, page 97-104, section 6.5.1
See ISO 1101, page 14, section 5.5

```
                    Start_entity
                    Toleranced_entity   : Array (1 to maxint) of
                            Profile_of_a_line_toleranced_entity (515)
                    Directrix           : Optional unit vector (101)
                    Tolerance           : Real
                    Application         : (I,B,O)
                    Primary_datum       : Optional Conditioned_datum (302)
                    Secondary_datum     : Optional Conditioned_datum (302)
                    Tertiary_datum      : Optional Conditioned_datum (302)
                    End_entity;
```

Attribute Descriptions:

   Toleranced_Entity: An array of entities to which
          the tolerance applies.

   Tolerance: The specified allowable deviation from
          the design nominal value.

   Application: An enumerated list that specifies the
          tolerance application to be I-inside,
          B-bilateral, or O-outside.

   Primary_datum: A conditioned_datum entity from which
          the dimension is measured. This primary datum
          is the most important datum relative to the
          tolerance. Required only when a unique plane
          cannot be calculated from the toleranced_entity.

   Secondary_datum: A conditioned_datum entity which is
          the second most important datum relative to the
          tolerance.

   Tertiary_datum: A conditioned_datum entity which is the
          third most important datum relative to the
          tolerance. With the primary and secondary data,
          it establishes a datum reference frame that exactly
          locates the toleranced feature.

   Directrix: Cross sections of the toleranced face taken in
          planes normal to the directrix establish the line
          profile that is toleranced. The directrix is
          required only if a datum is not included in the
          line tolerance entity.

(416) Profile of a surface

A profile of a surface tolerance specifies the distance between two
"ball-offset" surfaces located on equally on either side, or totally
on one side, of the design nominal feature. The toleranced feature
must lie between these surfaces. The tolerance is three-dimensional
and applies normal (perpendicular) to the true profile at all points.
Where a sharp corner is included, the tolerance zone extends to the
intersection of the boundary.
See ANSI Y14.5M 1982, page 97-104, section 6.5.1
See ISO 1101, page 14, section 5.5

```
                    Start_entity
                    Toleranced_entity   : Array (1 to maxint) of
                            Profile_of_a_surface_toleranced_entity (516)
                    Tolerance           : Real
                    Application         : (I,B,O)
                    Primary_datum       : Optional Conditioned_datum (302)
                    Secondary_datum     : Optional Conditioned_datum (302)
                    Tertiary_datum      : Optional Conditioned_datum (302)
                    End_entity;
```

Attribute Descriptions:

Toleranced_Entity: An array of entities to which
        the tolerance applies.

Tolerance: The specified allowable deviation from
        the design nominal value.

Application: An enumerated list that specifies the
        tolerance application to be I-inside,
        B-bilateral, or O-outside.

Primary_datum: A conditioned_datum entity from which
        the dimension is measured. This primary datum
        is the most important datum relative to the
        tolerance.

Secondary_datum: A conditioned_datum entity which is
        the second most important datum relative to the
        tolerance.

Tertiary_datum: A conditioned_datum entity which is the
        third most important datum relative to the
        tolerance.   With the primary and secondary data,
        it establishes a datum reference frame that exactly
        locates the toleranced feature.

(417) Straightness

Straightness tolerance defines the allowable deviation of a line or of line elements of a surface feature. The tolerance zones for line features are cylindrical, parallelepipedic and parallel planes. Straightness tolerance for surface features specify the distance between two parallel lines within which a linear element of the surface, in a specified direction, must lie. The linear element is a cross section of the surface in a plane parallel to the direction vector and normal to the surface. See figures 27,29 and 31 in the ISO 1101 standard, page 17.
See ANSI Y14.5M 1982, page 91-94, section 6.4.1
See ISO 1101, page 13, section 5.1

```
              Start_entity
              Toleranced_entity    : Array (1 to maxint) of
                                     Straightness_toleranced_entity (517)
              Direction            : Optional Unit_vector (101)
              Tolerance            : Real
              Material_condition   : (M, L, S)
              Cylindrical_zone     : Boolean
              Per_unit_length      : Real
              End_entity;
```

Attribute Descriptions:

Toleranced_Entity: An array of entities to which
     the tolerance applies.

Direction: A unit vector that specifies the
     straightness tolerance for linear surface
     elements. Required when the toleranced_entity
     is a surface. It must be parallel to the surface.

Tolerance: The specified allowable deviation from
     the design nominal value.

Material_condition: An enumerated list that indicates
     the material condition at which the tolerance
     applies: M - maximum material condition; L -
     least material condition; or S - regardless of
     feature size.

Cylindrical_zone: A boolean (true/false) flag that
     indicates that the tolerance value is the dia-
     meter of a cylindrical zone within which the
     axis or line must lie. If false, the zone is
     parallelepipedic or the space between two
     parallel lines or planes.

Per_unit_length: Specifies the linear distance
     within which the tolerance value applies. Used to
     prevent abrupt changes in the direction of the
     toleranced entity.

(418) Total runout

A total runout tolerance specifies the maximum allowable deviation of
position for all points on the toleranced feature during one complete
revolution of the feature about the datum axis, with relative axial
displacement of the measuring position. Where applied to surfaces
constructed around a datum axis, it is used to control the cumulative
variations of circularity, straightness, angularity, taper, profile of
a surface, and coaxiality. Where applied to surfaces constructed at
right angles to the datum axis, total runout controls perpendicularity
and flatness.
See ANSI Y14.5M 1982, page 109, section 6.7.2.2
See ISO 1101, page 22-23, section 5.12

Note that ANSI and ISO do not explicitly define total runout for
non-cylindrical surfaces. It is assumed that this is due to
mechanical measurement limitations.

```
          Start_entity
          Toleranced_entity    : Array (1 to maxint) of
                                 Total_runout_toleranced_entity (513)
          Tolerance            : Real
          Primary_datum        : Datum (301)
          Co-primary_datum     : Optional Datum (301)
          End_entity;
```

Attribute Descriptions:

Toleranced_Entity: An array of entities to which
    the tolerance applies.

Tolerance: The specified allowable deviation from
    the design nominal value.

Primary_datum: A datum entity from which
    the dimension is measured. This primary datum
    is the most important datum relative to the
    tolerance.

Co-primary_datum: An additional datum entity which
    establishes an axis with the primary datum.

(500) Angle toleranced entity

Angle toleranced entity is a point vector and a member from the class
edge (202), planar face (206) and size feature (303).

```
            Start_entity
            Toleranced_entity   : Face (203), edge (202),
                                    size_feature (303)
            Plane of measure    : Unit_vector (101)
            Toleranced_location : Array of 1:N of Point_vector
                                    (102)
            End_entity;
```

    Attribute Descriptions:

        Toleranced_Entity: An entity to which the tolerance
            applies.

        Plane of measure: Unit vector which specifies the
            normal to the plane in which the angle is
            measured.
            Note: The path vector must be normal to the
                ang-tol-ents(500) point vector and the
                ang-org(501) vector. The path must exist
                vector(500) and the vector(501) do not
                define a unique plane.

        Toleranced_location: A point_vector which specifies
            the reference orientation on the toleranced
            entity. It must lie on the toleranced entity.
            Multiple point_vectors apply to ruled surface
            toleranced_entities.

(501) Angle origin

Angle origin entity is an origin vector and a member of the class edge
(202), face (203) or datum (301).

```
            Start_entity
            Origin_entity       : Face (203), edge (202), or
                                    datum (301)
            Origin_vector       : Unit_vector (101)
            End_entity;
```

    Attribute Descriptions:

        Origin_entity: A face, edge, or datum that serves as
            the base of the calculated dimension.   It is
            the "from" entity of the directed dimension.

        Origin_vector: A unit_vector which specifies the
            base orientation. The origin vector must be
            contained in the origin entity and at the
            location of the toleranced entity.

(502) Location toleranced entity

Location toleranced entity is a location coordinate and a member from
the class of edge (202), face (203), location tolerance qualified form
feature (533), size feature (303), vertex (201).

```
                    Start_entity
                    Toleranced_entity    : Face, edge, vertex,
                                             size_feature, or
                                             location_tolerance_
                                               qualified_form_feature
                    Toleranced_location : Coordinate (107)
                    End_entity;
```

    Attribute Descriptions:

        Toleranced_Entity: An entity to which the tolerance
                applies.

        Toleranced_location: A coordinate which specifies
                the reference location on the toleranced entity.
                The coordinate must lie on the toleranced entity.

(503) Location origin

Location origin is an origin coordinate and a member from the class
edge (202), face (203), datum (301), vertex (201).

```
                    Start_entity
                    Origin_entity        : Face, edge, vertex, or
                                             datum
                    Origin_location      : Coordinate (107)
                    End_entity;
```

    Attribute Descriptions:

        Origin_entity: An entity that serves as the base of
                the calculated dimension. It is the "from"
                entity of the directed dimension.

        Origin_location: A coordinate which specifies the base
                position. The coordinate must lie on the origin
                entity.

(504) Location path

A location path is a class of entities:

```
            Class of: unit_vector (101)
                      curve (105)
            End_class;
```

It is used to specify the path along which a dimension measure is calculated. A curve instance must contain but not necessarily end with the toleranced coordinate point(s).

(505) Size toleranced entity

Size toleranced entity is a class of entities:

```
            Class of: size_feature (303)
                      circular edge (204)
                      cylindrical face (205)
                      size_tolerance_
                                qualified_form_feature (519)
            End_class;
```

Each of these entities has a symmetric set of geometric entities which is toleranced as to size of the symmetric elements.

(506) Angularity toleranced entity

Angularity toleranced entity is a class of entities:

```
            Class of: face (203)
                      edge (202)
                      size_feature (303)
                      angularity_tolerance_
                                qualified_form_feature (520)
            End_class;
```


(507) Circular runout toleranced entity

Circular runout toleranced entity is a class of entities:

```
            Class of: circular edge (204)
                      runout_face (208)
                      circular_runout_tolerance_
                                qualified_form_feature (520)
            End_class;
```

(508) Circularity toleranced entity

Circularity toleranced entity is a class of entities:

                Class of: circular_face (205)
                          circular_edge (204)
                          circularity_tolerance_
                                    qualified_form_feature (524)
                End_class;

(509) Concentricity toleranced entity

Concentricity toleranced entity is a class of entities:

                Class of: circular_face (205)
                          circular_edge (204)
                          concentricity_tolerance_
                                    qualified_form_feature (521)
                End_class;

(510) Cylindricity toleranced entity

Cylindricity toleranced entity is a class of entities:

                Class of: cylindrical_face (207)
                          cylindricity_tolerance_
                                    qualified_form_feature (525)
                End_class;

(511) Flatness toleranced entity

Flatness toleranced entity is a class of entities:

                Class of: planar_face (206)
                          flatness_tolerance_
                                    qualified_form_feature (522)
                End_class;

(512) Parallelism toleranced entity

Parallelism toleranced entity is a class of entities:

                Class of: face (203)
                          edge (202)
                          size_feature (303)
                          parallelism_tolerance_
                                    qualified_form_feature (527)
                End_class;

(513) Perpendicularity toleranced entity

Perpendicularity toleranced entity is a class of entities:

                    Class of: face (203)
                              edge (202)
                              size_feature (303)
                              perpendicularity_tolerance_
                                        qualified_form_feature (526)
                    End_class;

(514) Position toleranced entity

Position toleranced entity is a class of entities:

                    Class of: size_feature (303)
                              position_tolerance_
                                        qualified_form_feature (520)
                    End_class;

(515) Profile-of-a-line toleranced entity

Profile of a line toleranced entity is a class of entities:

                    Class of: face (203)
                              planar_edge (209)
                              profile_of_a_line_tolerance_
                                        qualified_form_feature (531)
                    End_class;

(516) Profile-of-a-surface toleranced entity

Profile of a surface toleranced entity is a class of entities:

                    Class of: face (203)
                              profile_of_a_surface_tolerance_
                                        qualified_form_feature (532)
                    End_class;

(517) Straightness toleranced entity

Straightness toleranced entity is a class of entities:

                    Class of: face (203)
                              edge (202)
                              size_feature (303)
                              straightness_tolerance_
                                        qualified_form_feature (523)
                    End_class;

(518) Total runout toleranced entity

Total runout toleranced entity is a class of entities:

                     Class of: runout_face (208)
                               total_runout_tolerance_
                                            qualified_form_feature (530)
                     End_class;

(519) Size tolerance qualified form feature

Size tolerance qualified form feature is a class of entities:

                     Class of: Hole (305)
                     End_class;

(520) Position tolerance qualified form feature

Position tolerance qualified form feature is a class of entities:

                     Class of: Hole (305)
                     End_class;

(521) Concentricity tolerance qualified form feature

Concentricity tolerance qualified form feature is a class of entities:

                     Class of: Hole (305)
                     End_class;

(522) Flatness tolerance qualified form feature

A flatness tolerance qualified form feature is a class of entities
which exhibit the properties of a plane in the region of interest.

                     Class of: <null>
                     End_class;

(523) Straightness tolerance qualified form feature

Straightness tolerance qualified form feature is a class of entities:

                     Class of: Hole (305)
                     End_class;

(524) Circularity tolerance qualified form feature

Circularity tolerance qualified form feature is a class of entities:

                    Class of: Hole (305)
                    End_class;

(525) Cylindricity tolerance qualified form feature

Cylindricity tolerance qualified form feature is a class of entities:

                    Class of: Hole (305)
                    End_class;

(526) Perpendicularity tolerance qualified form feature

Perpendicularity tolerance qualified form feature is a class of entities:

                    Class of: Hole (305)
                    End_class;

(527) Parallelism tolerance qualified form feature

Parallelism tolerance qualified form feature is a class of entities:

                    Class of: Hole (305)
                    End_class;

(528) Angularity tolerance qualified form feature

Angularity tolerance qualified form feature is a class of entities:

                    Class of: Hole (305)
                    End_class;

(529) Circular runout tolerance qualified form feature

Circular runout tolerance qualified form feature is a class of entities:

                    Class of: Hole (305)
                    End_class;

(530) Total runout tolerance qualified form feature

Total runout tolerance qualified form features is a class of entities:

                    Class of: Hole (305)
                    End_class;

(531) Profile of a line tolerance qualified form feature

Profile of a line tolerance qualified form features is a class of entities:

                    Class of: <null>
                    End_class;

(532) Profile of a surface tolerance qualified form feature

Profile of a surface tolerance qualified form features is a class of entities:

                    Class of: <null>
                    End_class;

(533) Location tolerance qualified form feature

Location tolerance qualified form features is a class of entities:

                    Class of: Hole (305)
                    End_class;

DATA MODELLING CONVENTIONS
31 OCTOBER 85

This model should be interpreted as an IDEF1 model with the
following conventions and assumptions:

1. An arc is used below or above the subject entity through a number of
   it's relationships to denote an exclusive-or. This means that only
   one relationship of those thorugh which the arc passes can exist for
   a given instance of the subject entity.

2. An unlabeled relationship that has parentheses on it near the
   independant entity denotes an "equivalence" relationship that is
   read "can be/is." This relationship exists primarily between class
   entities and the members of the class.

3. On "diamond" relationships a range indicator of the form N:M is
   placed beside the diamond. This indicates the minimum and maximum
   number of entities which can be referenced in the relationship.

4. There is a distinction made between two types of relationships and
   two types of entities. The two types of relationships are the
   typical IDEF1 relationship with a verb-like label and the
   relationship described in item 3. The two types of entities are
   "class" entities and the typical IDEF1 entities which have
   attributes. Class entities have no attributes and are used to group
   entities that are similar in nature.

5. An asterisk that is used in place of an entity number indicates that
   the definition of that entity is outside the scope of the model.
   The entity, in this case, serves as a link to other models.

MAR 2, 1986
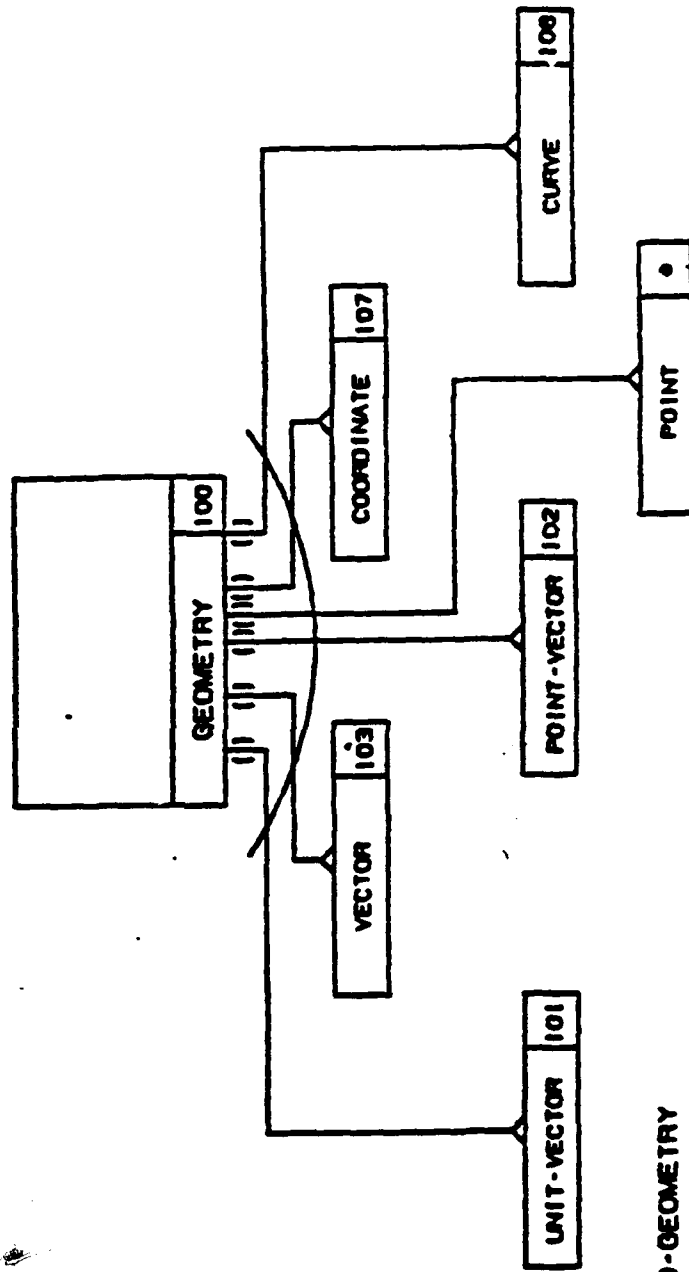
L51-100-GEOMETRY

MAR 28 1986
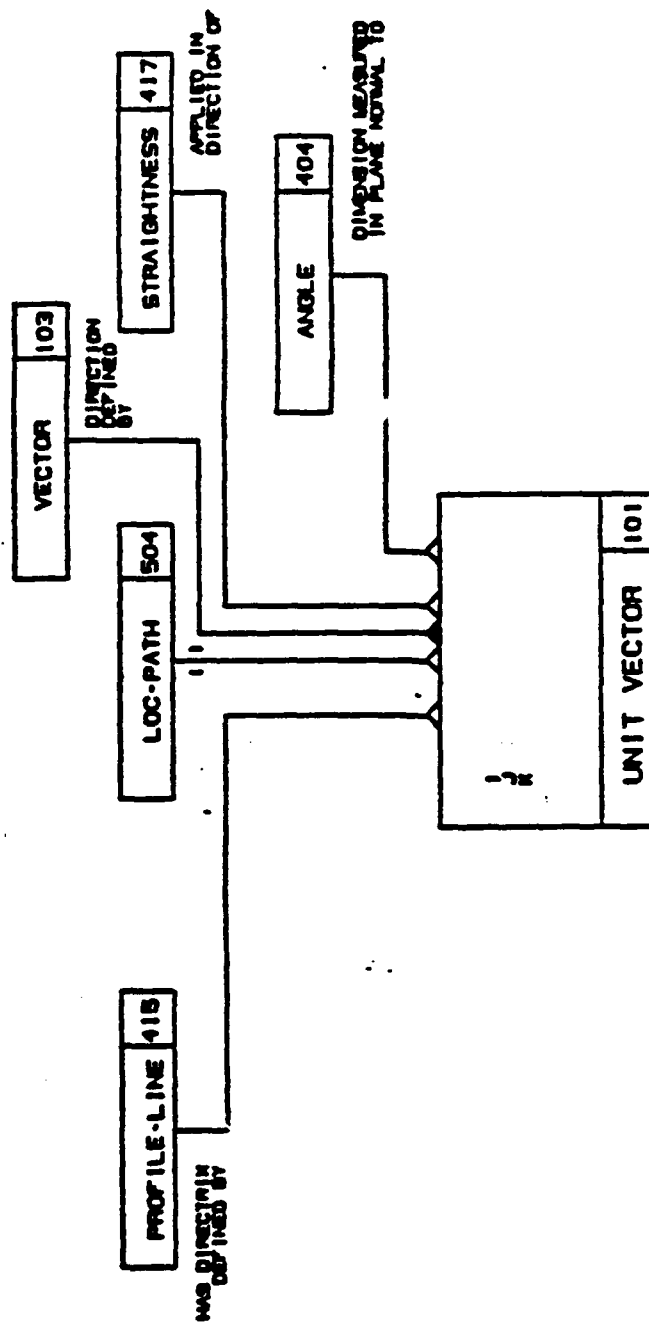
L18-101-UNIT VECTOR
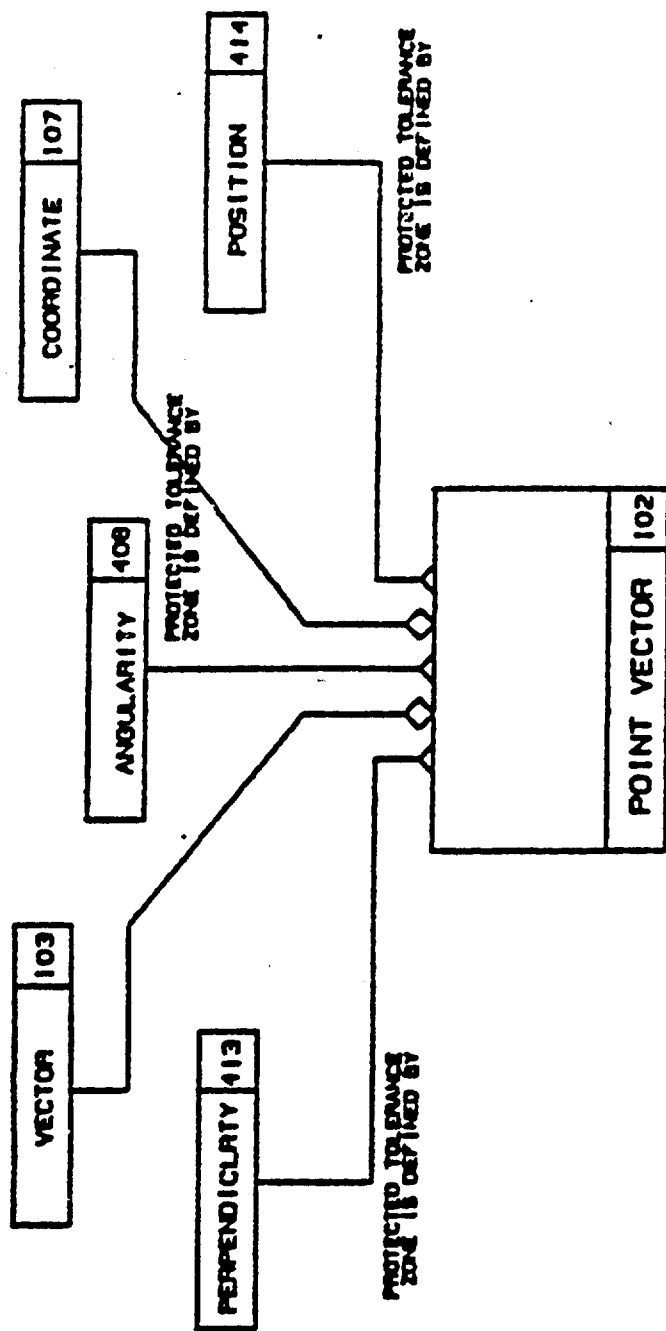
COORDINATE 107

POSITION 414

ANGULARITY 408

VECTOR 103

PERPENDICULRTY 413

POINT VECTOR 102

PROJECTED TOLERANCE ZONE IS DEFINED BY

PROJECTED TOLERANCE ZONE IS DEFINED BY

PROJECTED TOLERANCE ZONE IS DEFINED BY

PROJECTED TOLERANCE ZONE IS DEFINED BY

L15-102-POINT VECTOR

L4-108-CURVE

LOC-ORIGIN 503

DIMENSION
MEASURED FROM
MUST LIE ON

COORDINATE 107

LOCATES

O-N

POINT-VECTOR 102

X-Y-Z

LOC-TOL-ENTS 502

TOLERANCED TO MUST LIE ON
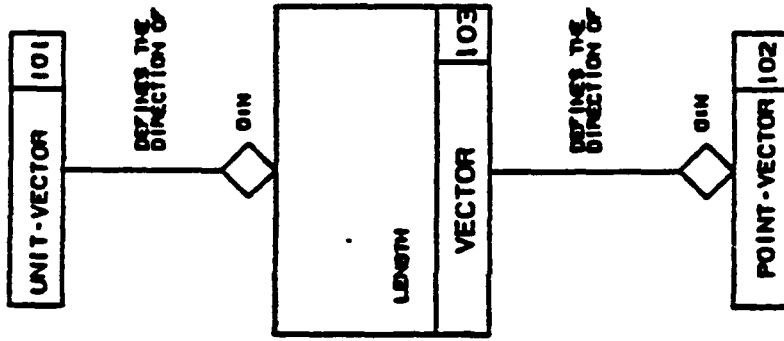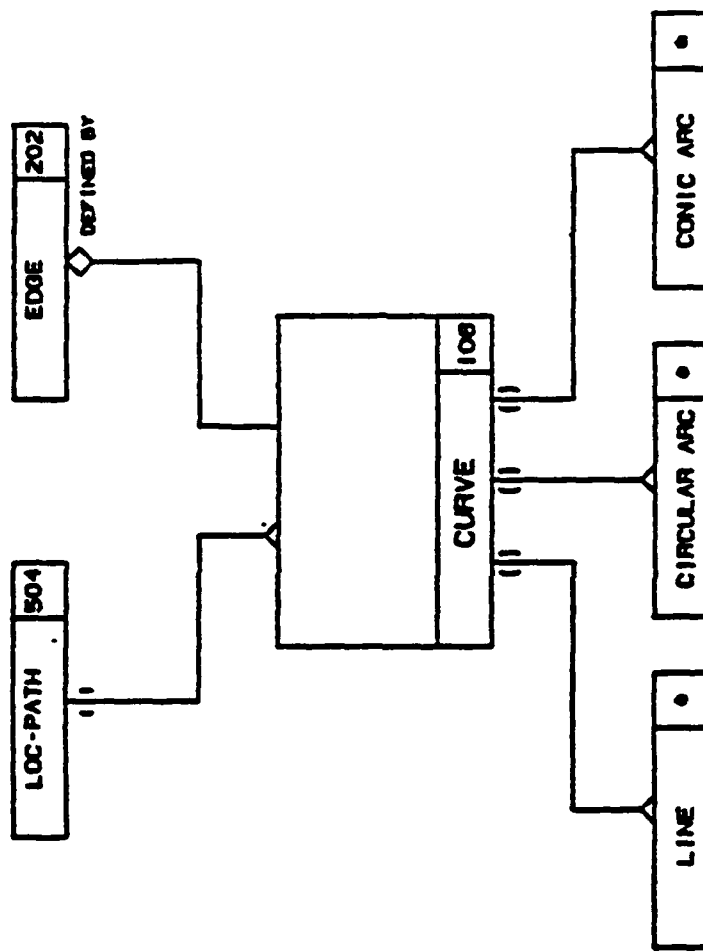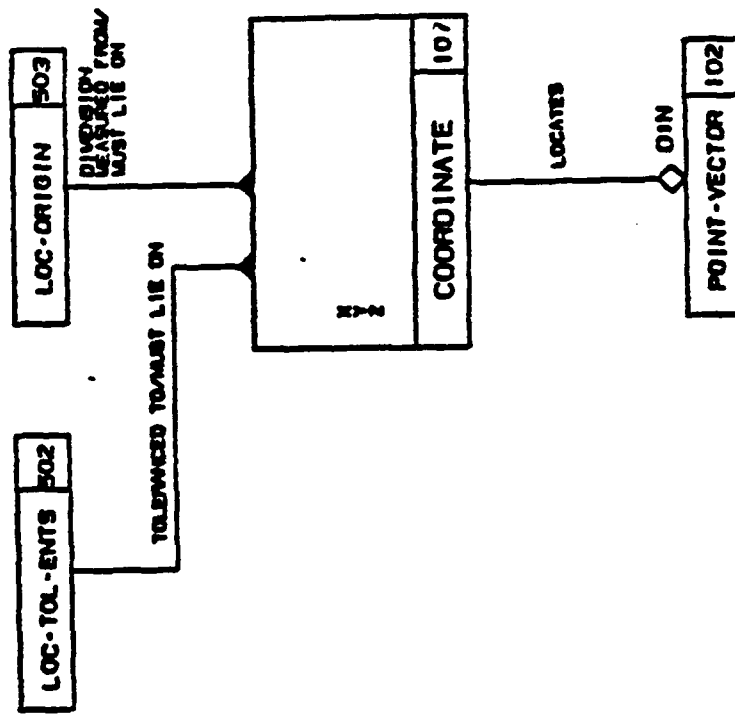
L52-107-COORDINATE

MAR 26 1986
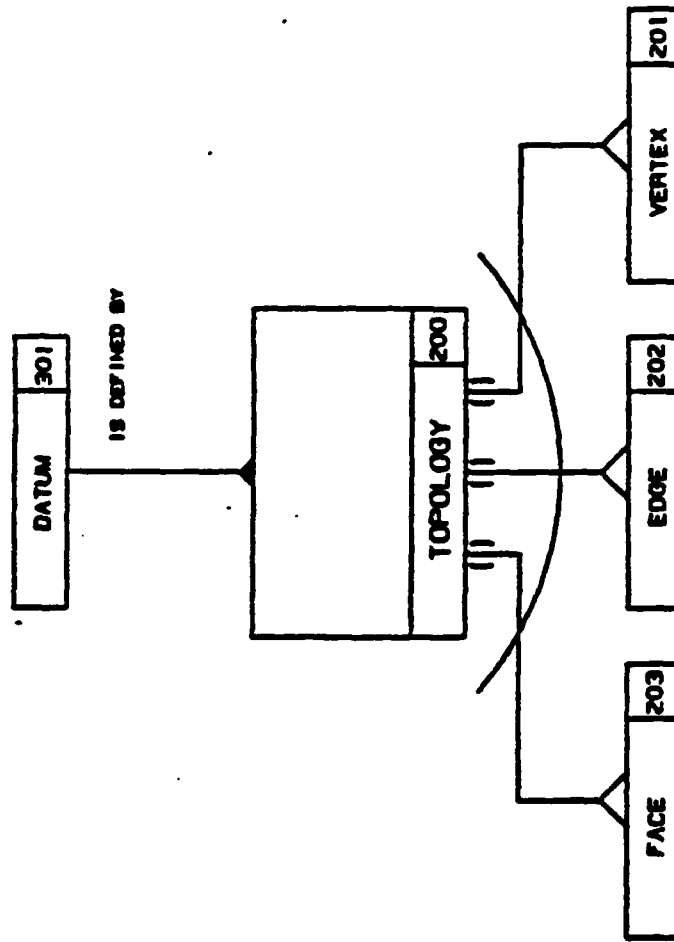
DATUM 301

IS DEFINED BY

TOPOLOGY 200

VERTEX 201

EDGE 202

FACE 203

A TOPOLOGY IS EITHER TO HAVE UNDERLYING, OR HAVING GEOMETRY.

L8-200-TOPOLOGY

POINT

LOCATES

POS-TOL-ENTS 514

EDGE 202

STARTED/ENDED BY

LOC-ORIGIN 503

LOC-TOL-ENTS 502

VERTEX 201
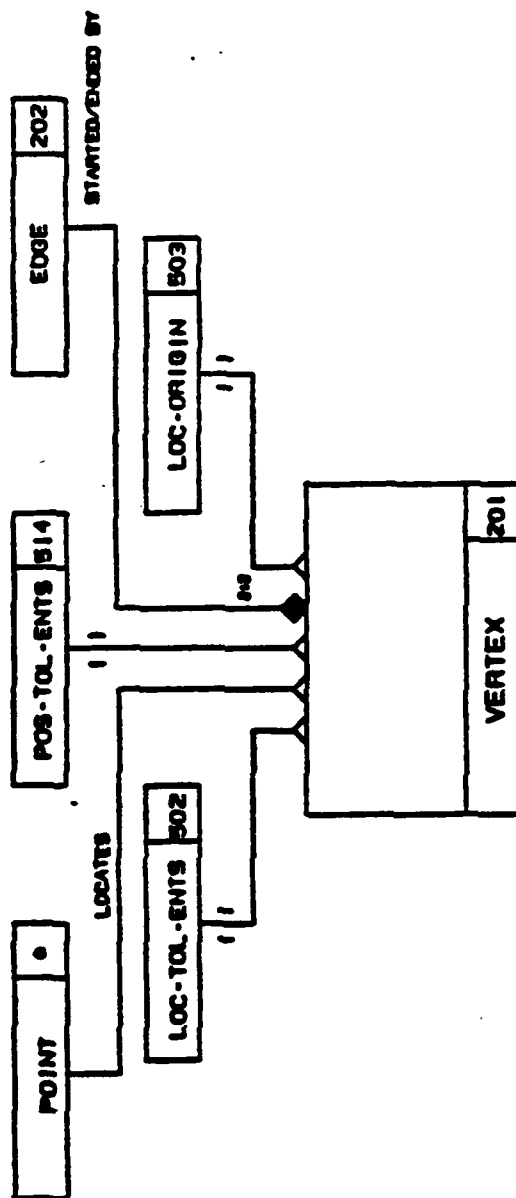
L14-201-VERTEX

MAR 26 1986

L17-202-EDGE

L18-203-FACE

PLANAR EDGE 209

CIRC-TOL-ENTS 506

CIRC-TOL-ENTS 506

CONC-TOL-ENTS 509

CRUN-TOL-ENTS 507
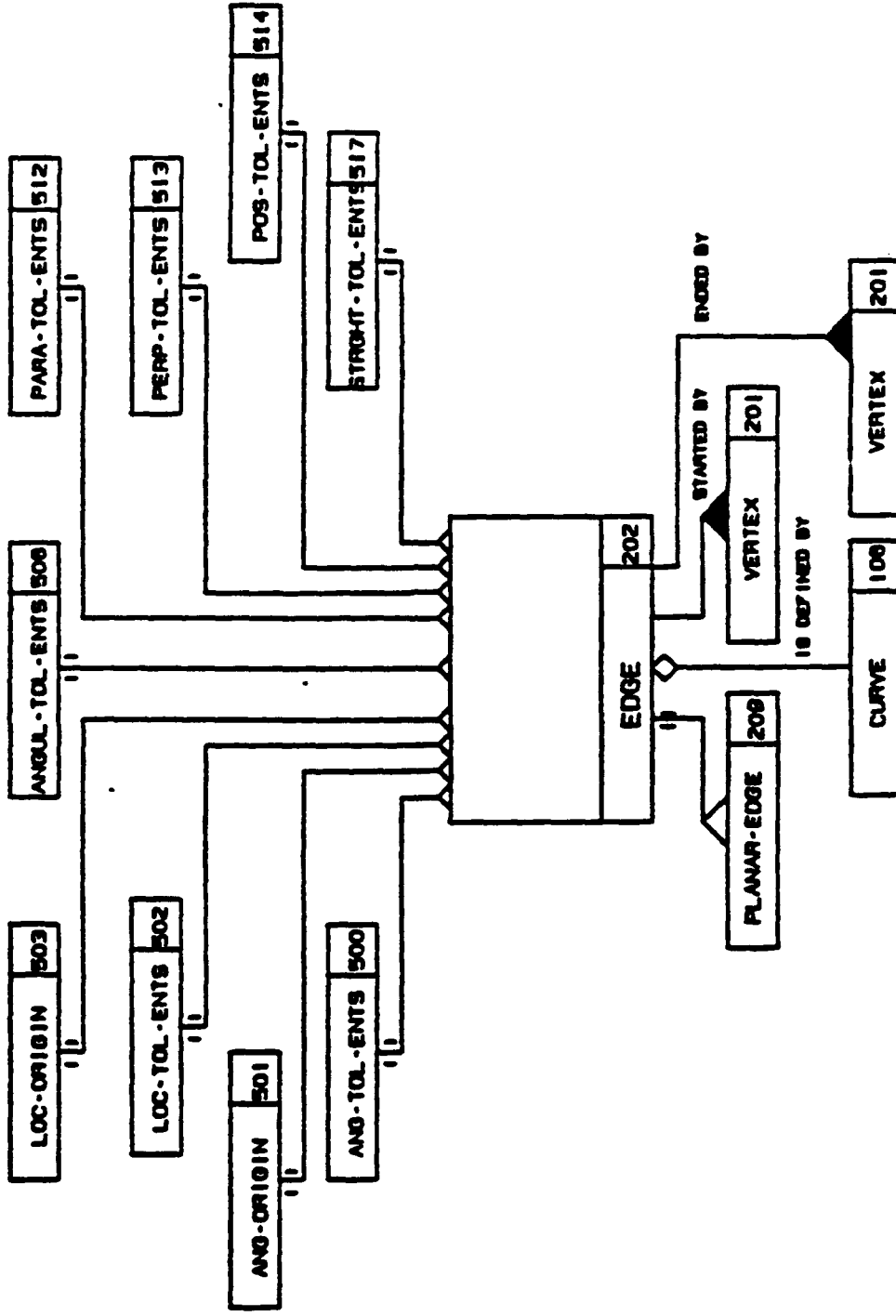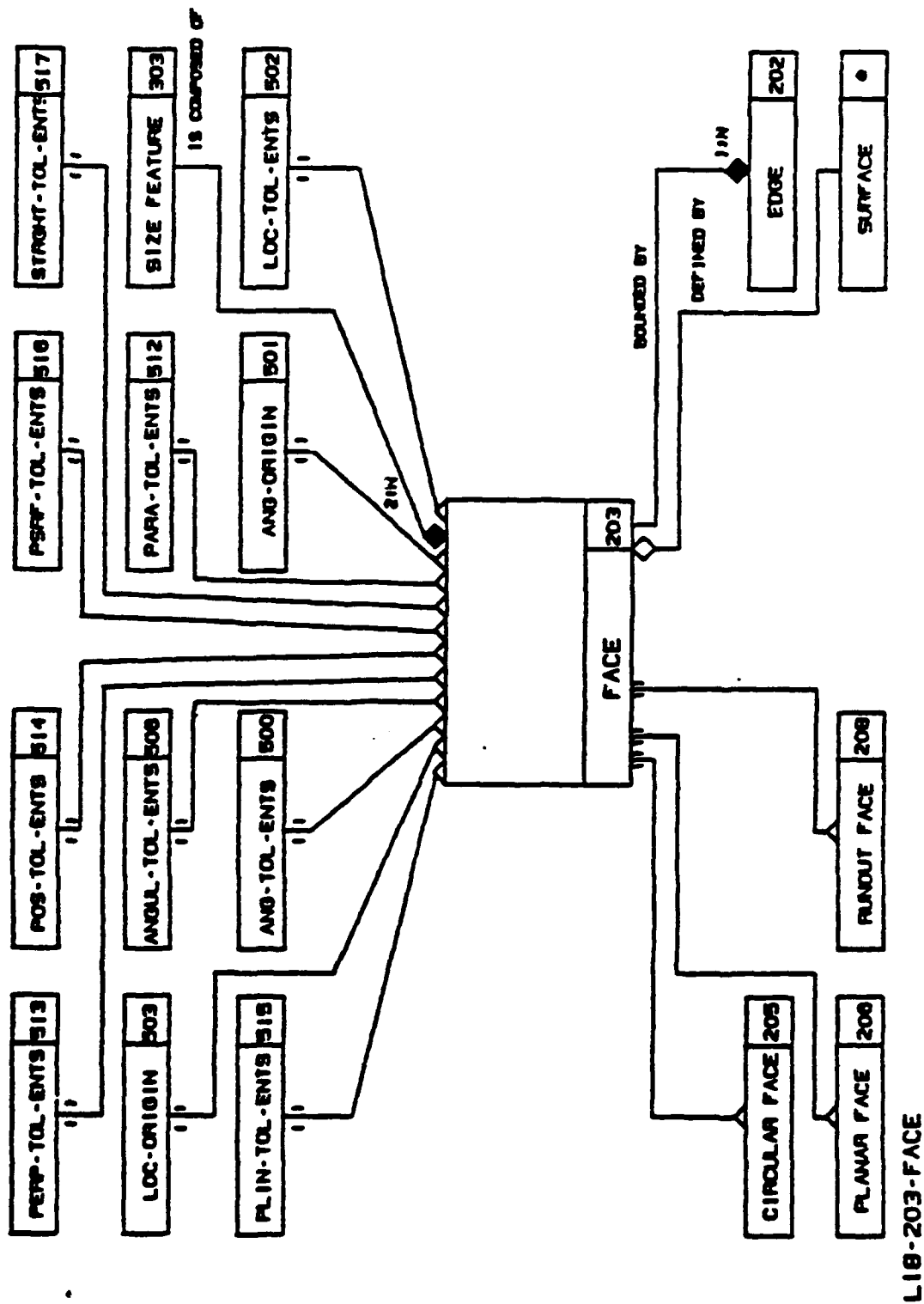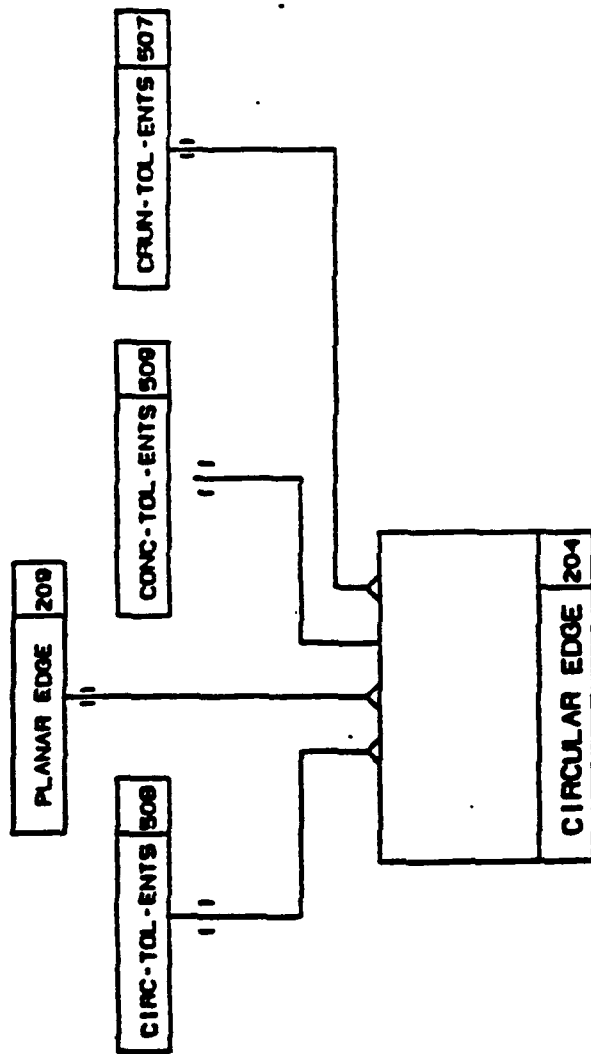
CIRCULAR EDGE 204

L12-204-CIRCULAR EDGE

MAR 26 1986

FACE 203

CONC-TOL-ENTS 509

CIRC-TOL-ENTS 508

CIRCULAR FACE 205

CYLINDRICAL FACE 207
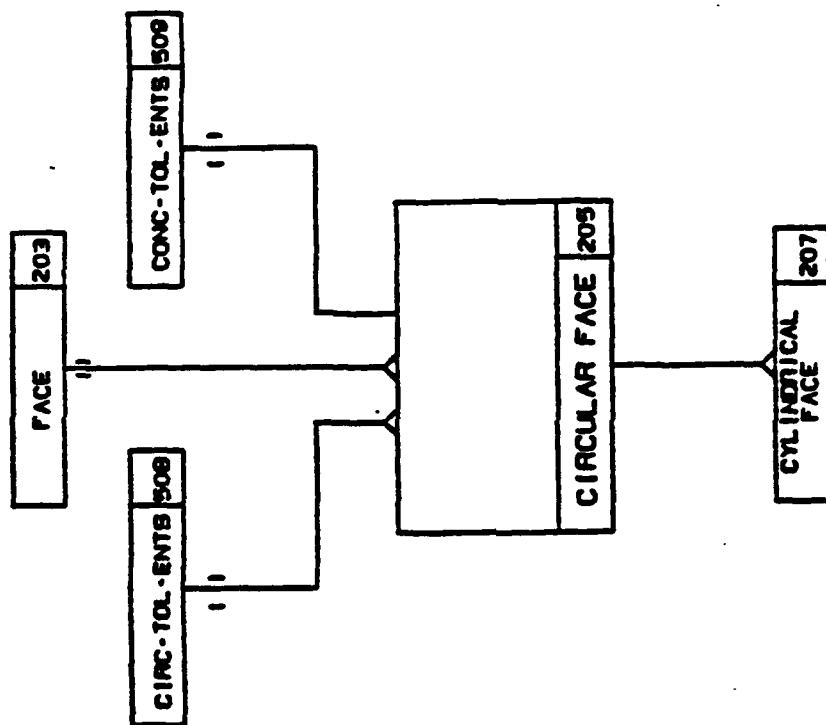
L13-205-CIRCULAR FACE

MAR 26 1986

L9-206-PLANAR FACE

CYL-TOL-ENTS 510   CIRCULAR FACE 205

CYLINDRICAL FACE 207

L10-207-CYLINDRICAL FACE

FACE | 203

TRUN·TOL·ENTS | 518

DRUN·TOL·ENTS | 507
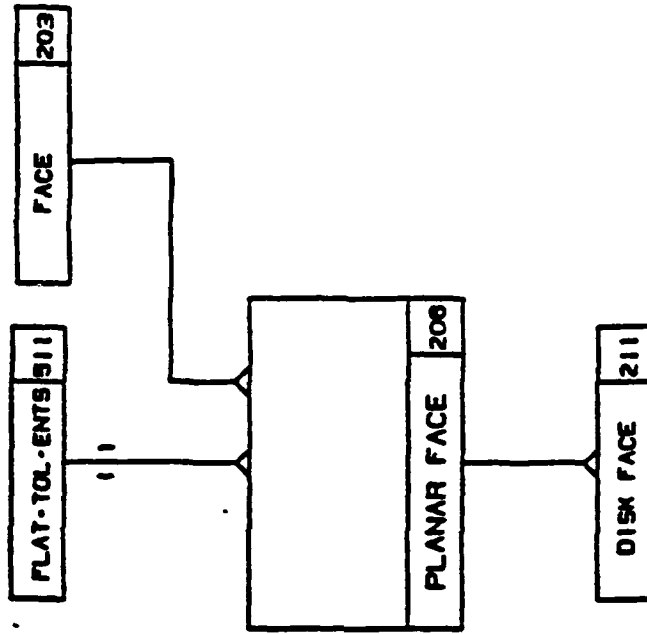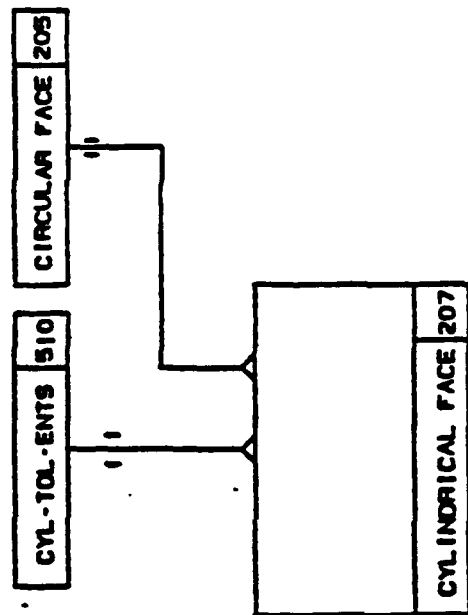
RUNOUT FACE | 208

DISK FACE | 211

CYLINDRICAL FACE | 207
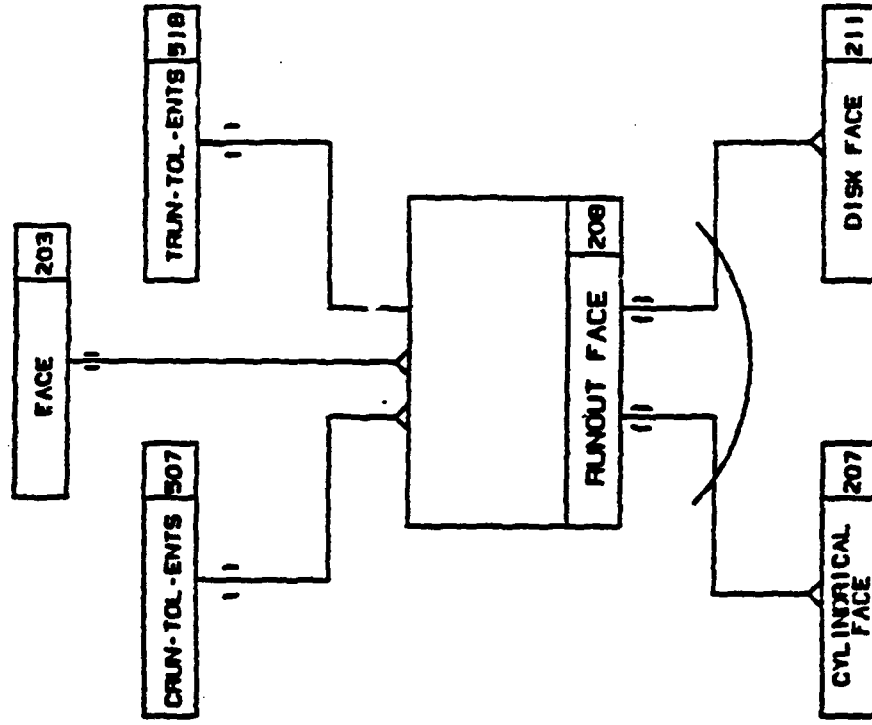
L11-208-RUNOUT FACE

MAR 26 1986

LB-208-PLANAR EDGE

RUNOUT FACE | 208
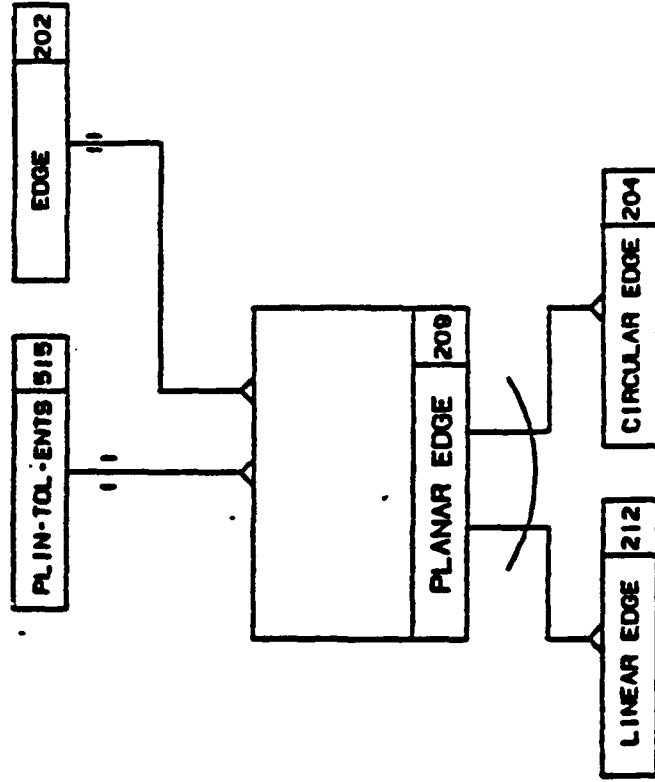
DISK FACE | 211

L53-211-DISK FACE

PLANAR EDGE 209
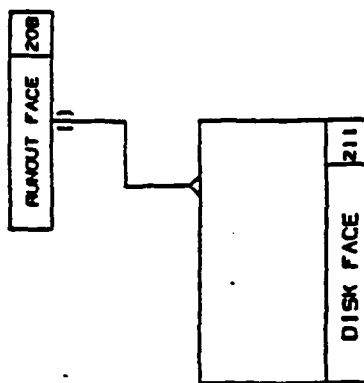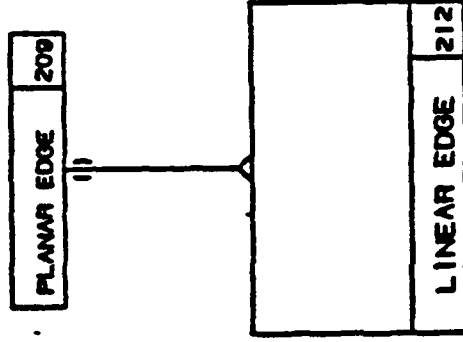
LINEAR EDGE 212

L54-212-LINEAR EDGE

MAR 26 1986

FEATURE 300

FORM-FEATURE 304
VERTEX 201
SIZE FEATURE 303
EDGE 202
FACE 203
DATUM 301

L48-300-FEATURE

MAR 26 1986

MAR 26 1986

L28-302-CONDITIONED DATUM

MAR 26 1986
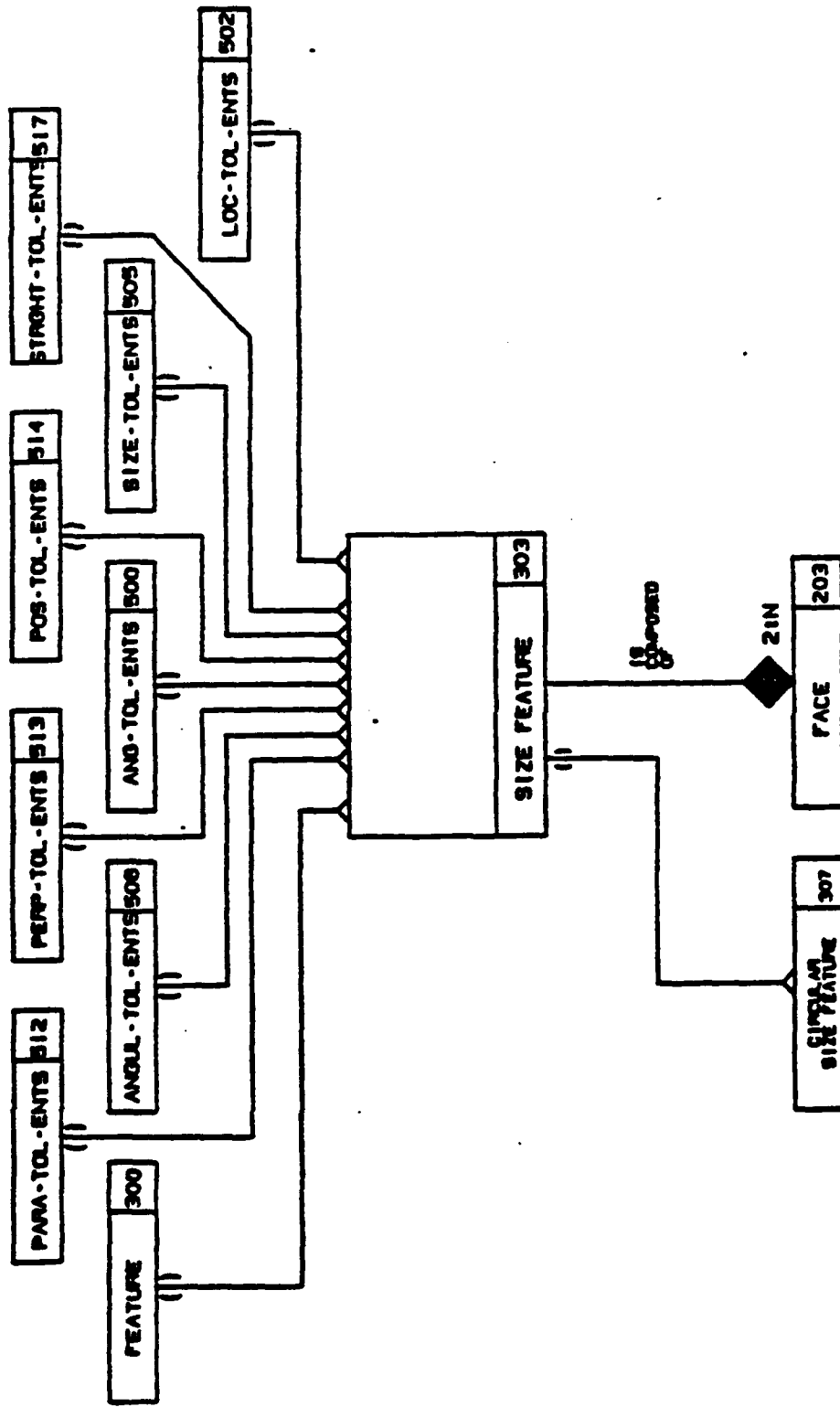
L2B-303-SIZE FEATURE
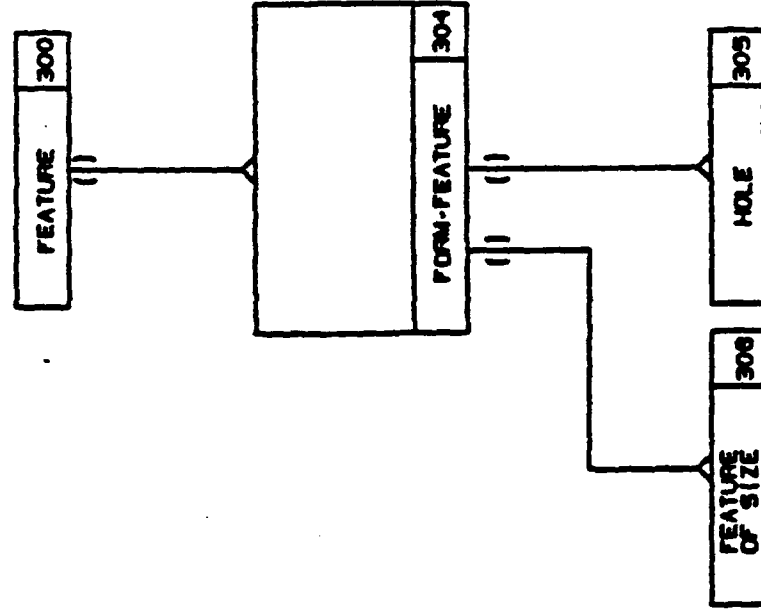
L-47-304-FORM FEATURE

MAR 26 1986

FEATURE OF SIZE — 306

(1)

HOLE — 305

L49·305·HOLE

FORM-FEATURE 304

FEATURE OF SIZE 306

HOLE 305
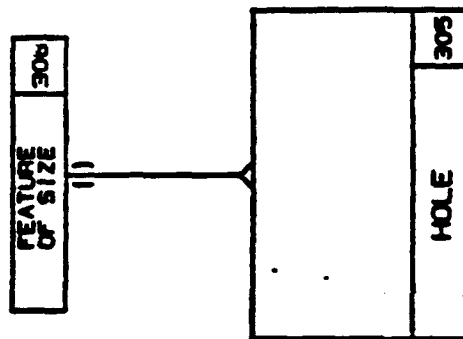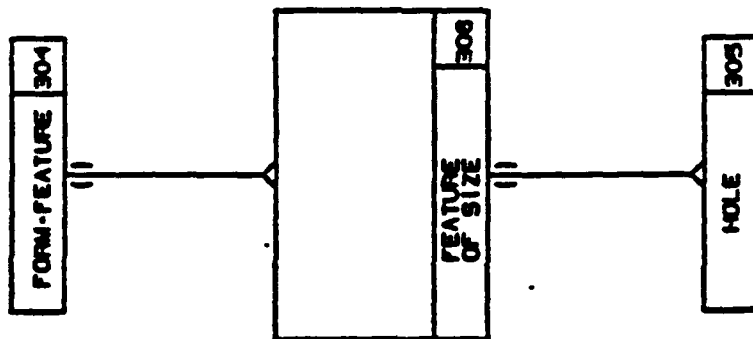
L50-300-FEATURE OF SIZE

MAR 26 1986

TOLERANCE 400

GEOM - TOL 102

COORD - TOL 101

L1-400-TOLERANCE

TOLERANCE 400

COORD-TOL 401

SIZE 405

ANGLE 404
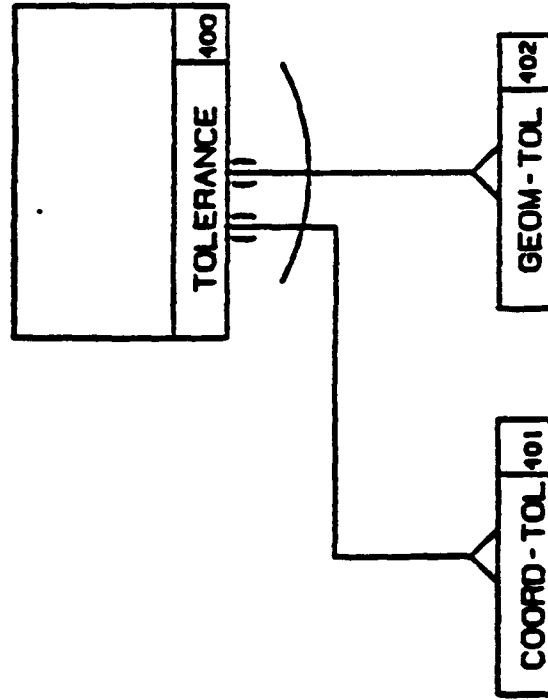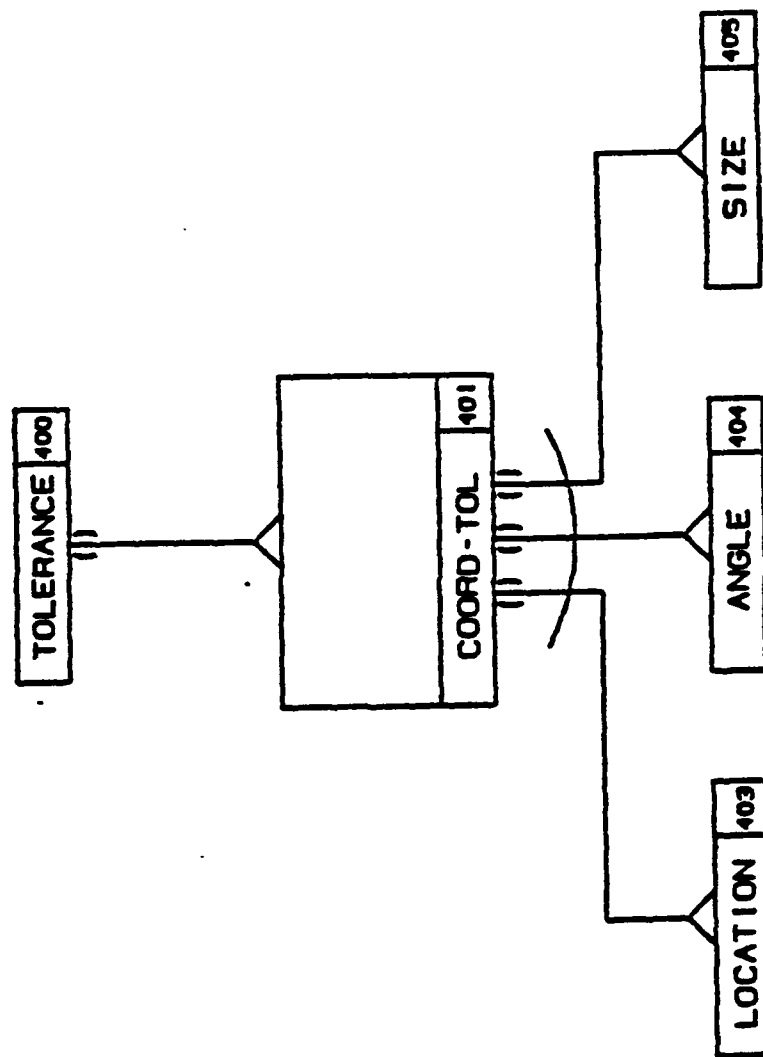
LOCATION 403

L2-401-COORDINATE TOLERANCE

MAR 26 1986

L12-4D2-GEOMETRIC TOLERANCE

MAR 26 1986

LB-403-LOCATION

MAR 26 1986

COORD - TOL 401

ANGLE

BASIC
ANG-TOL
VALUE
466

404

COORDINATE 107

UNIT-VECTOR 101

ANG-ORIGIN 501

ANG-TOL-ENTS 500

① EXPLAIN RELATIONSHIP TO THE
CALCULATED VECTORS AT THE COORDINATE

② COORDINATE(S) MUST LIE ON INTERSECTION
OF 500 AND 501. THE VECTORS USED TO CALCULATE
THE ANGLE ARE THE VECTORS TANGENT TO 500
AND 501 AT THE COORDINATE AND PERPENDICULAR
TO THE R&R VECTOR.

HAS INTERSECTION POINT 8
DEFINED BY

RIGHT-
HAND-RULE
ESTABLISHED BY

USED AS
BASIS FOR
DIMENSION

TOLERANCES

1IN

1IN

L3-404-ANGLE

L10-405-SIZE

MAR 26 1986

L13-406-ANGULARITY
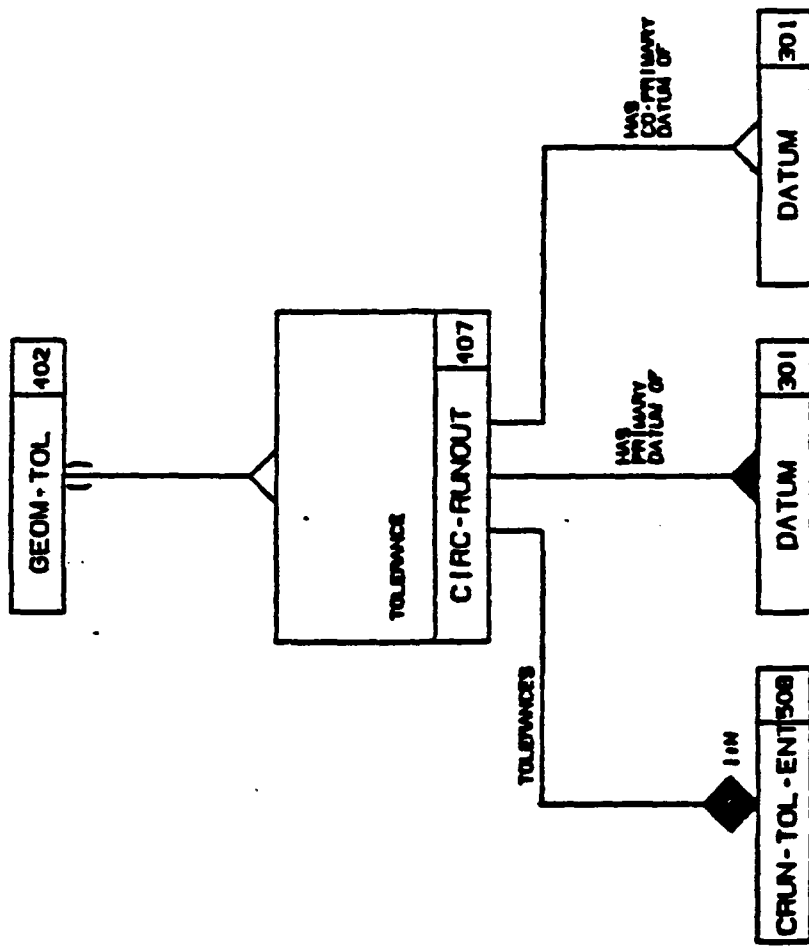
L14-407-CIRCULAR RUNOUT

MAR 26 1986
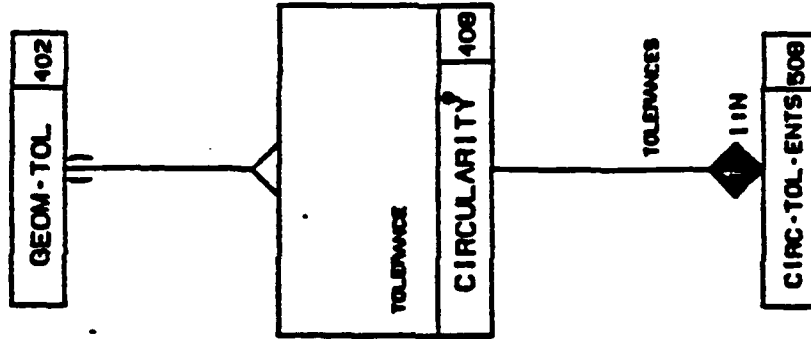
L22-408-CIRCULARITY
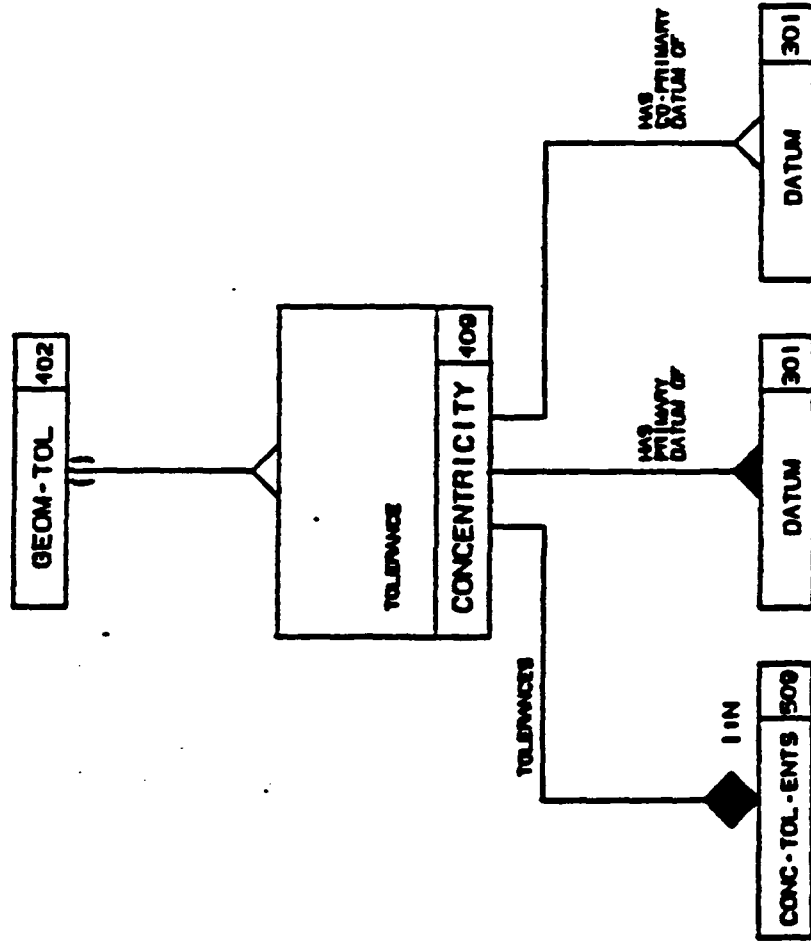
L15-409-CONCENTRICITY

MAR 26 1986

L16-410-CYLINDRICITY

GEOM-TOL 402

TOLERANCE
MATERIAL CONDITION
PER-UNIT-CORRELATION

FLATNESS 411

TOLERANCES

IIN

FLAT-TOL-ENTS 511
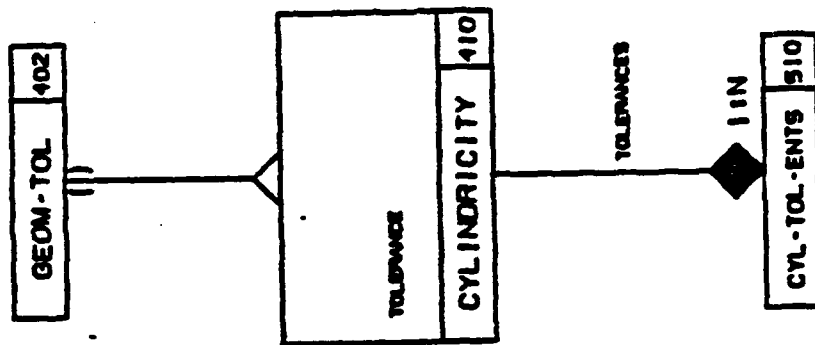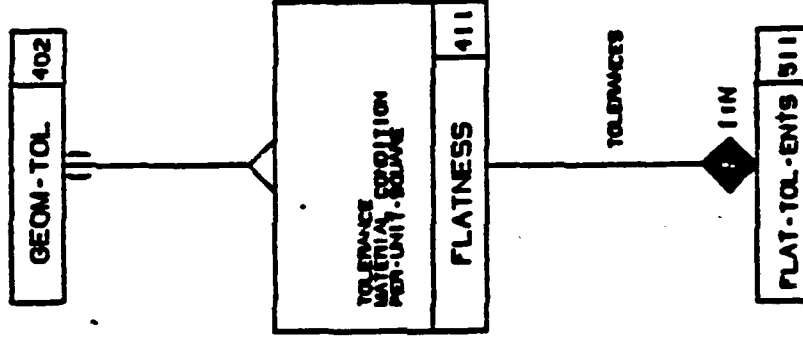
L17-411-FLATNESS

MAR 26 1986
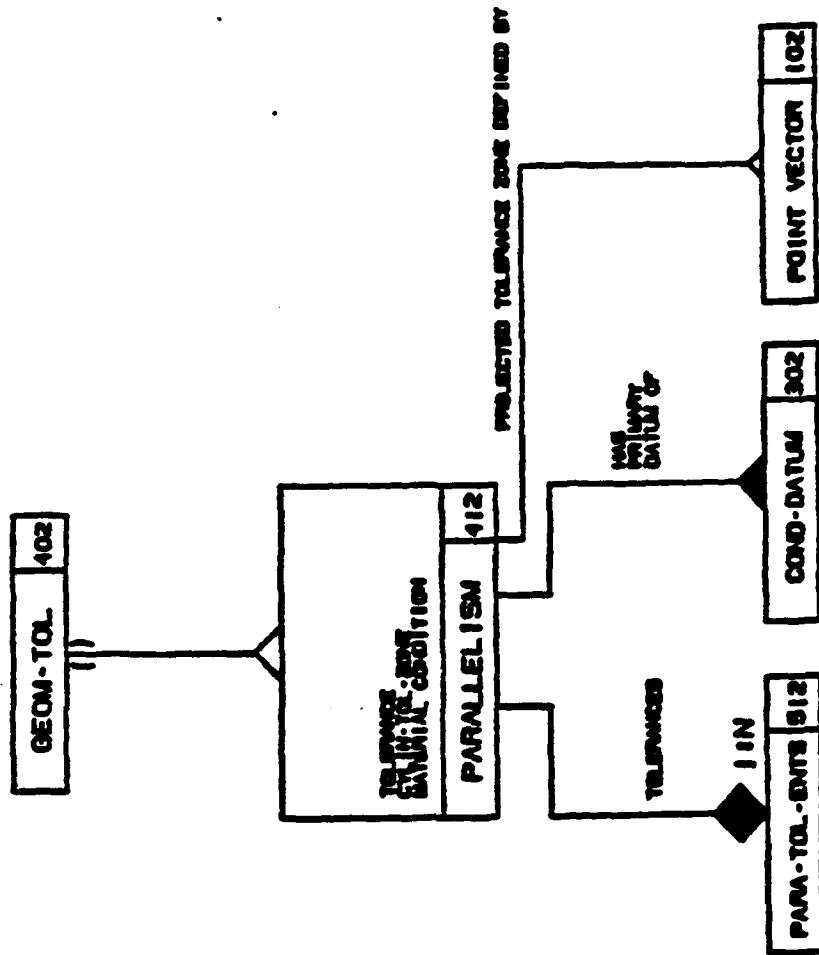
L18-412-PARALLELISM

MAR 2 6 1986
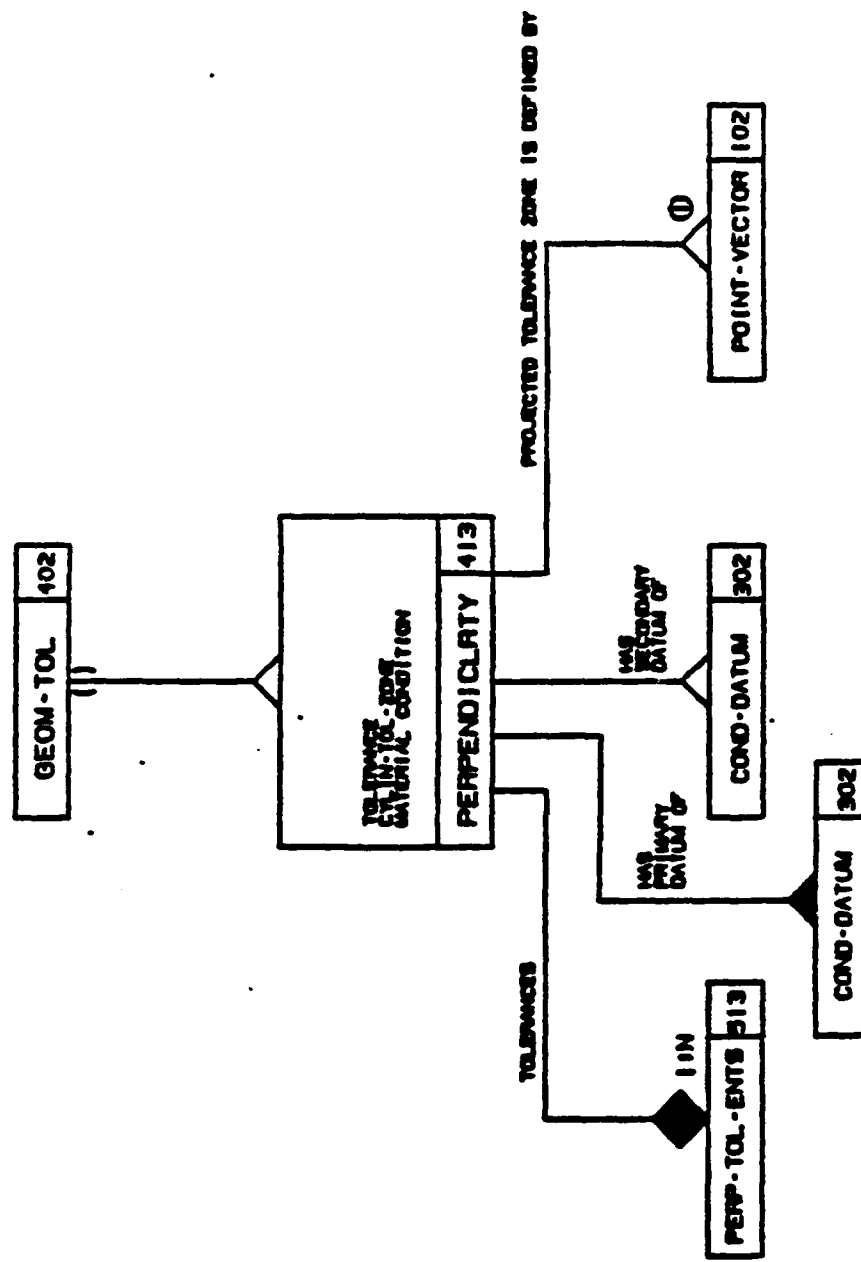
L20-413-PERPENDICULARITY

MAR 26 1986

GEOM-TOL 402

TOLERANCE
ZONE
MATERIAL CONDITION

POSITION 414

HAS TERTIARY DATUM OF

HAS SECONDARY DATUM OF

HAS PRIMARY DATUM OF

TOLERANCES

POINT-VECTOR 102

COND-DATUM 302

COND-DATUM 302

COND-DATUM 302

POS-TOL-ENTS 314

PROJECTED TOLERANCE ZONE IS DEFINED BY
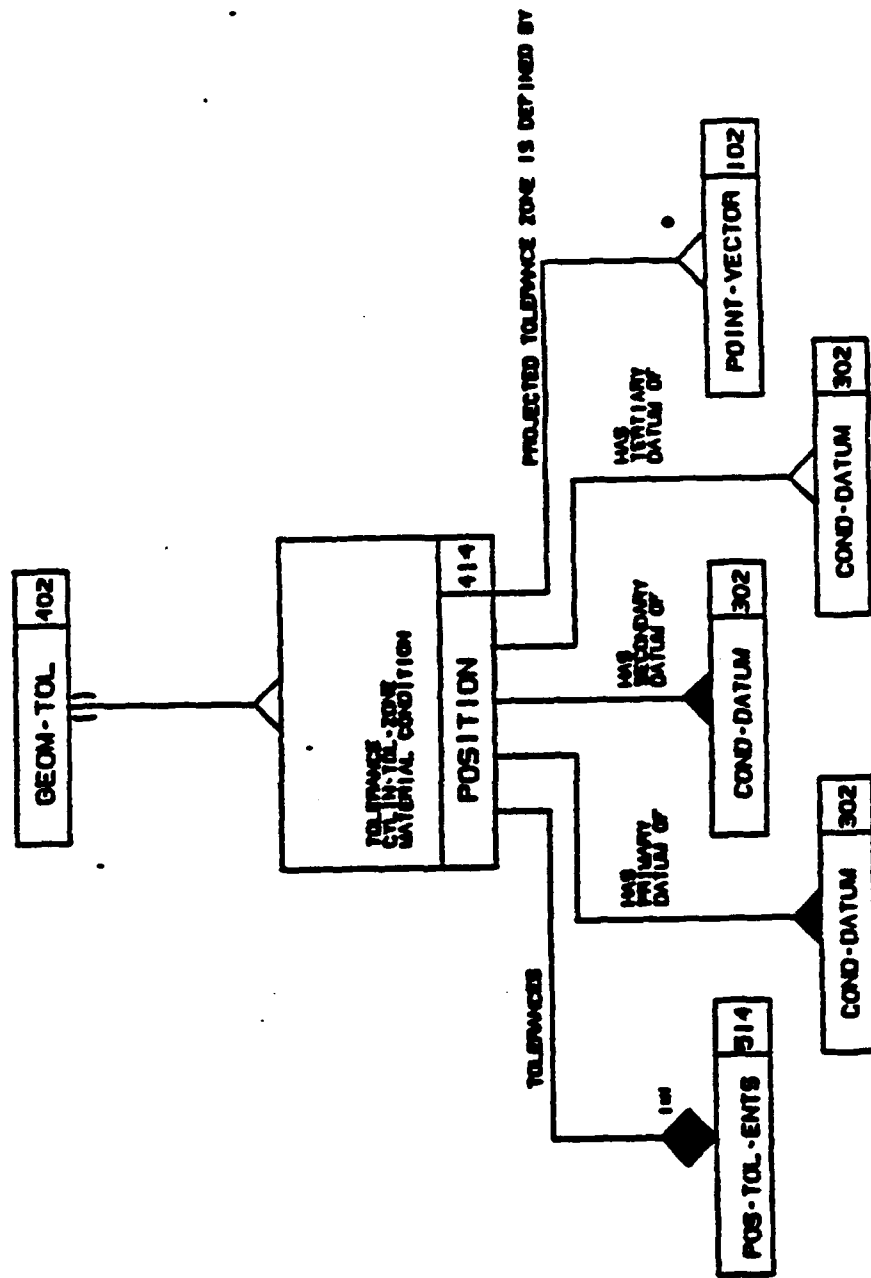
• LENGTH OF VECTOR ESTABLISHES EXTENT OF ZONE

L21-414-POSITION

MAR 26 1986
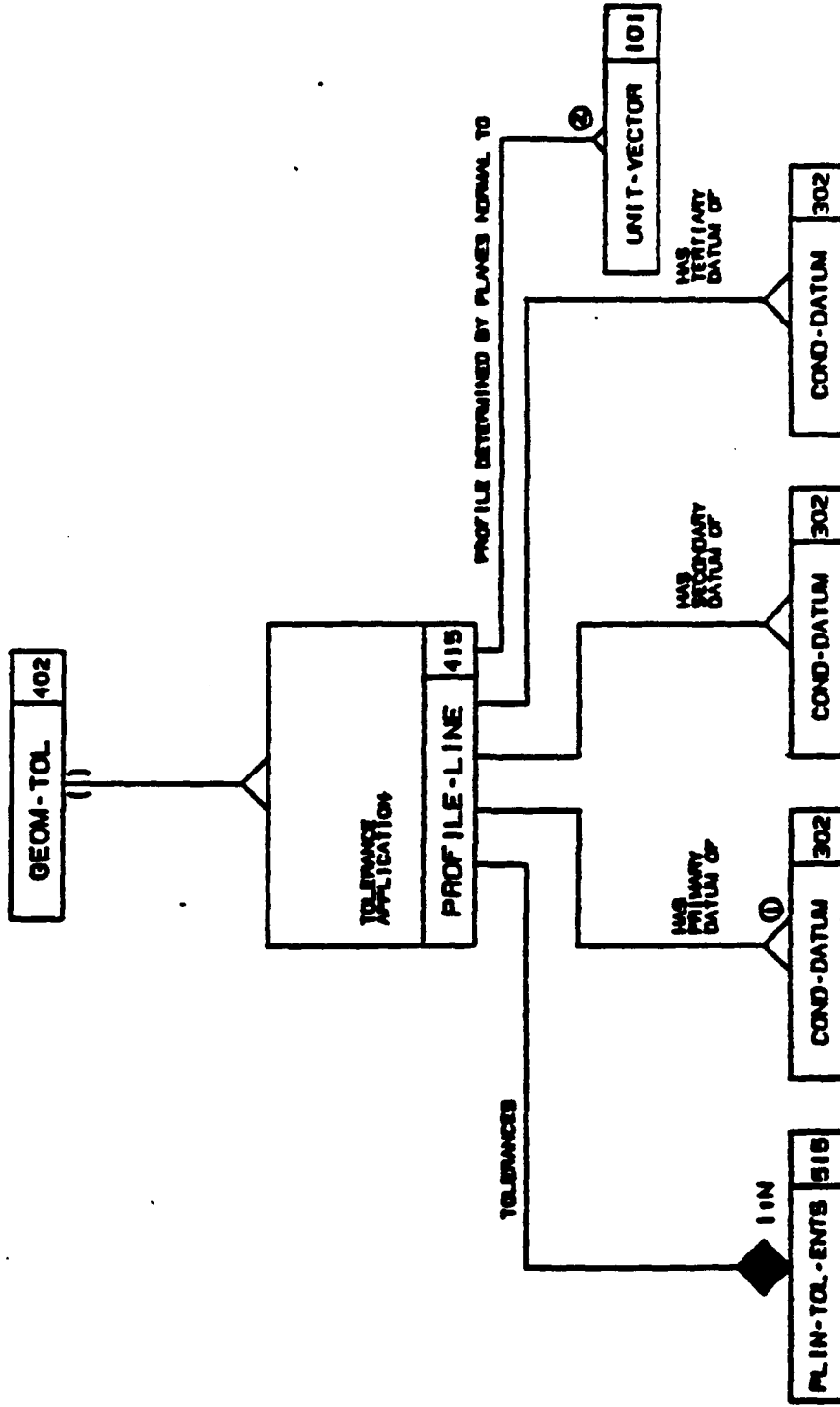
L18-415-PROFILE OF A LINE

① REQUIRED MEAN UNIQUE PLANE CANNOT BE CALCULATED FROM THE TOLERANCED CURVE.

② FOR REGIONS OF A TOLERANCED FACE (AREA) IN THE PROFILE-LINE THAT THE VECTOR ESTABLISHED UNIT VECTOR IS REQUIRED FROM THE IS A SPECIFIED UNIT VECTOR MUST BE PERPENDICULAR TO SPECIFIED DATUMS

L24-410-PROFILE OF A SURFACE

MAR 26 1986

GEOM-TOL 402

TOLERANCE
CYL.IN:TOL.ZONE
MATERIAL.CONDITION
PER-UNIT.LENGTH

STRAIGHTNESS 417

APPLIED IN DIRECTION OF

UNIT-VECTOR 101

TOLERANCES

1:N

STRGHT-TOL-ENTS 517

① MUST BE INTERPRETED ONLY WHEN TOLERANCED ENTITY IS SURFACE.
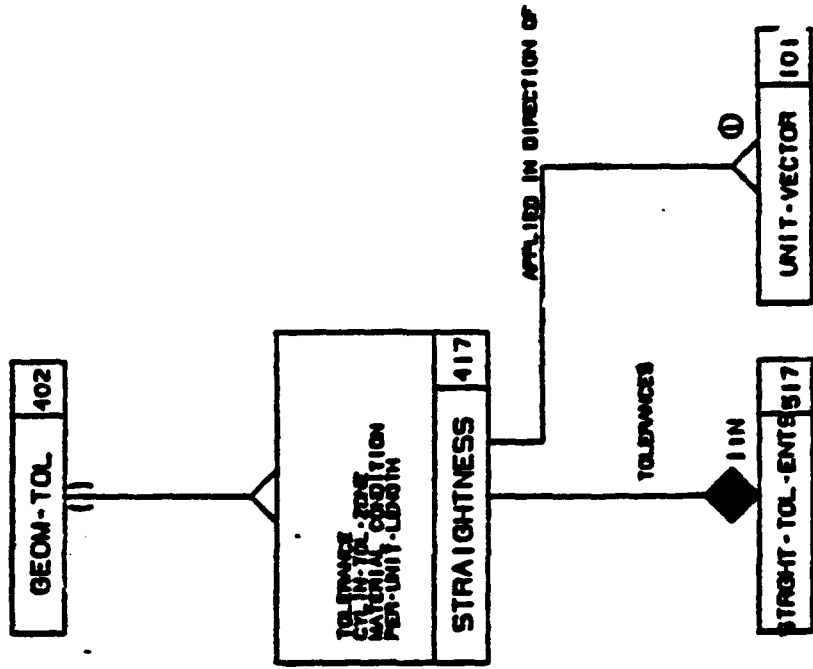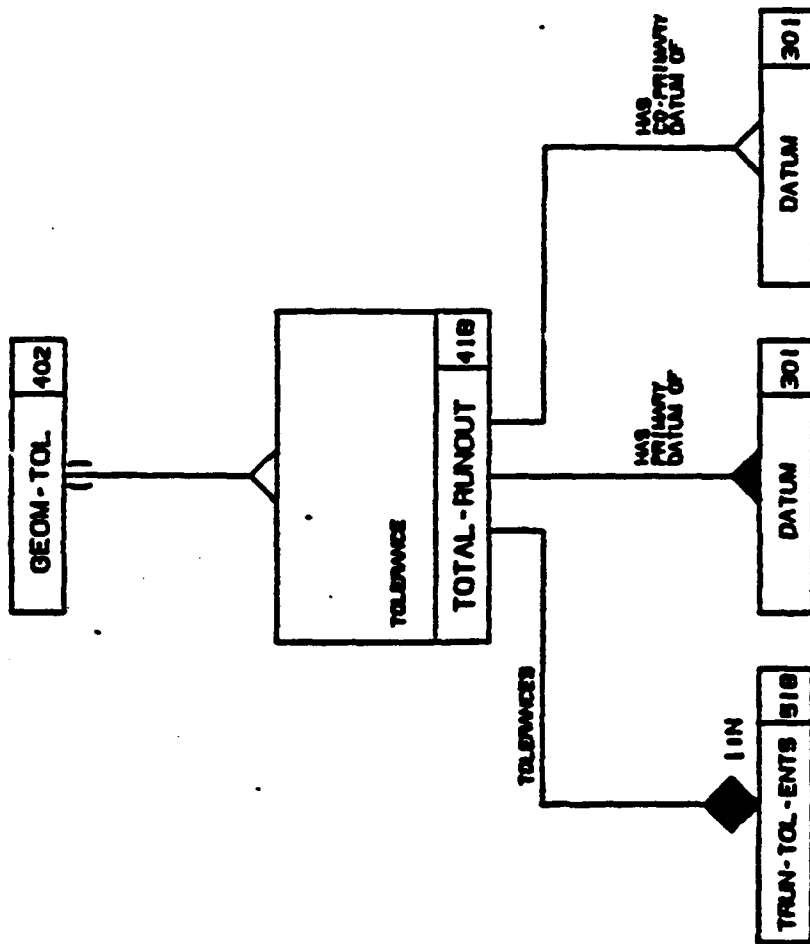
L23-417-STRAIGHTNESS

L25-418-TOTAL RUNOUT

L4-500-ANGLE TOLERANCED ENTITIES

MAR 26 1986

ANGLE 404

USES AS
ORIGIN
DIMENSION

ANG-ORIGIN 501

DATUM 301

EDGE 202

FACE 203
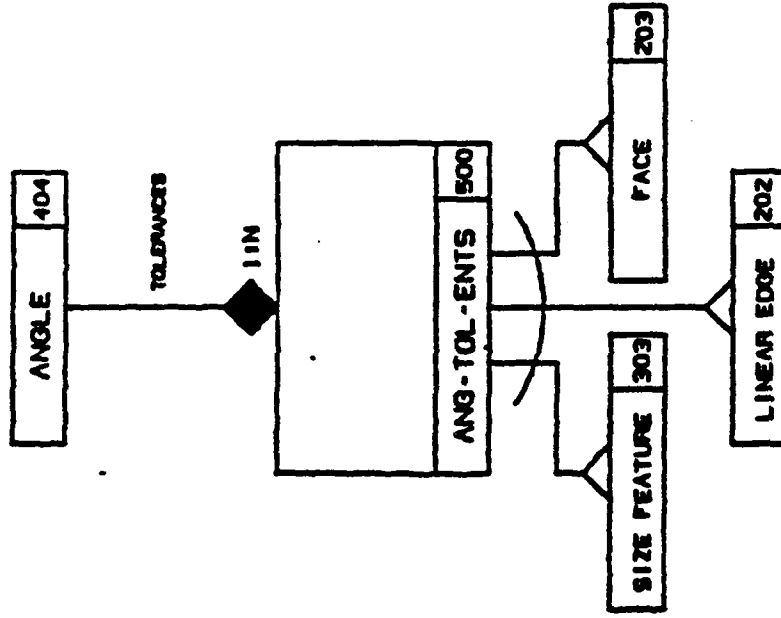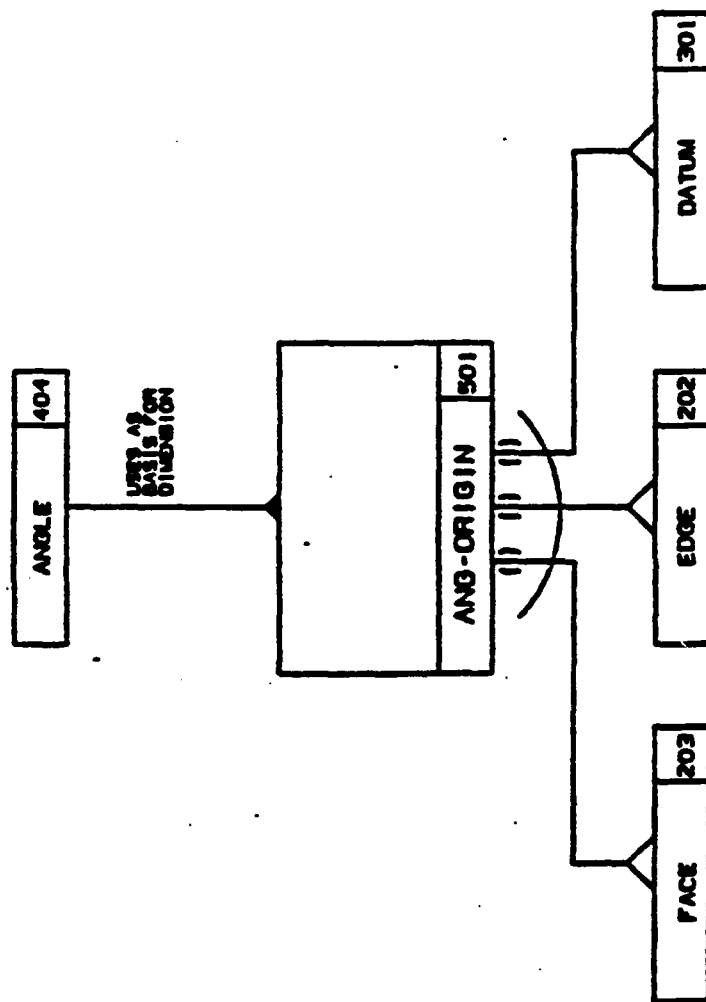
LS-501-ANGLE ORIGIN

MAR 26 1986
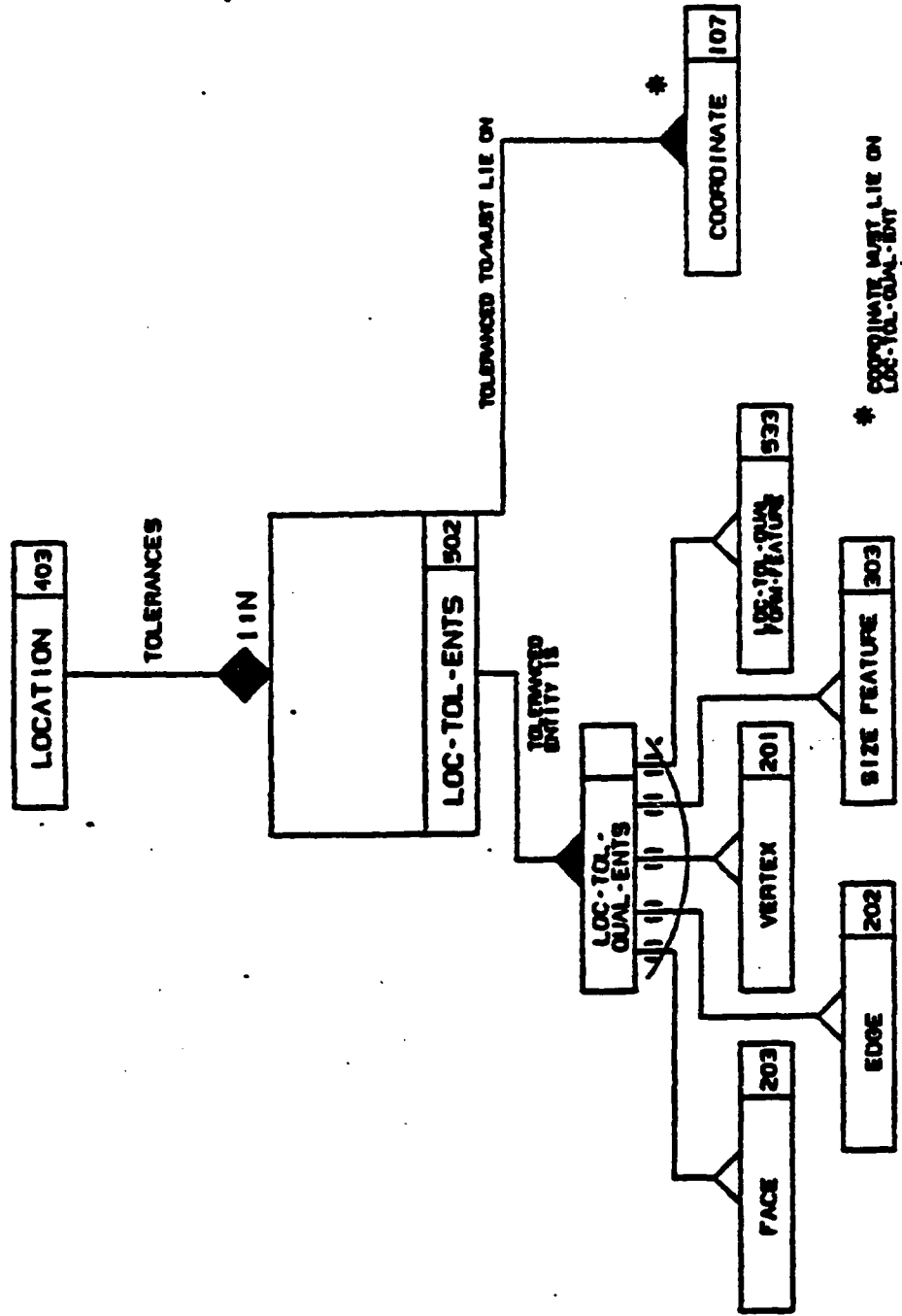
MAR 26 1986

L7-502-LOCATION TOLERANCED ENTITIES
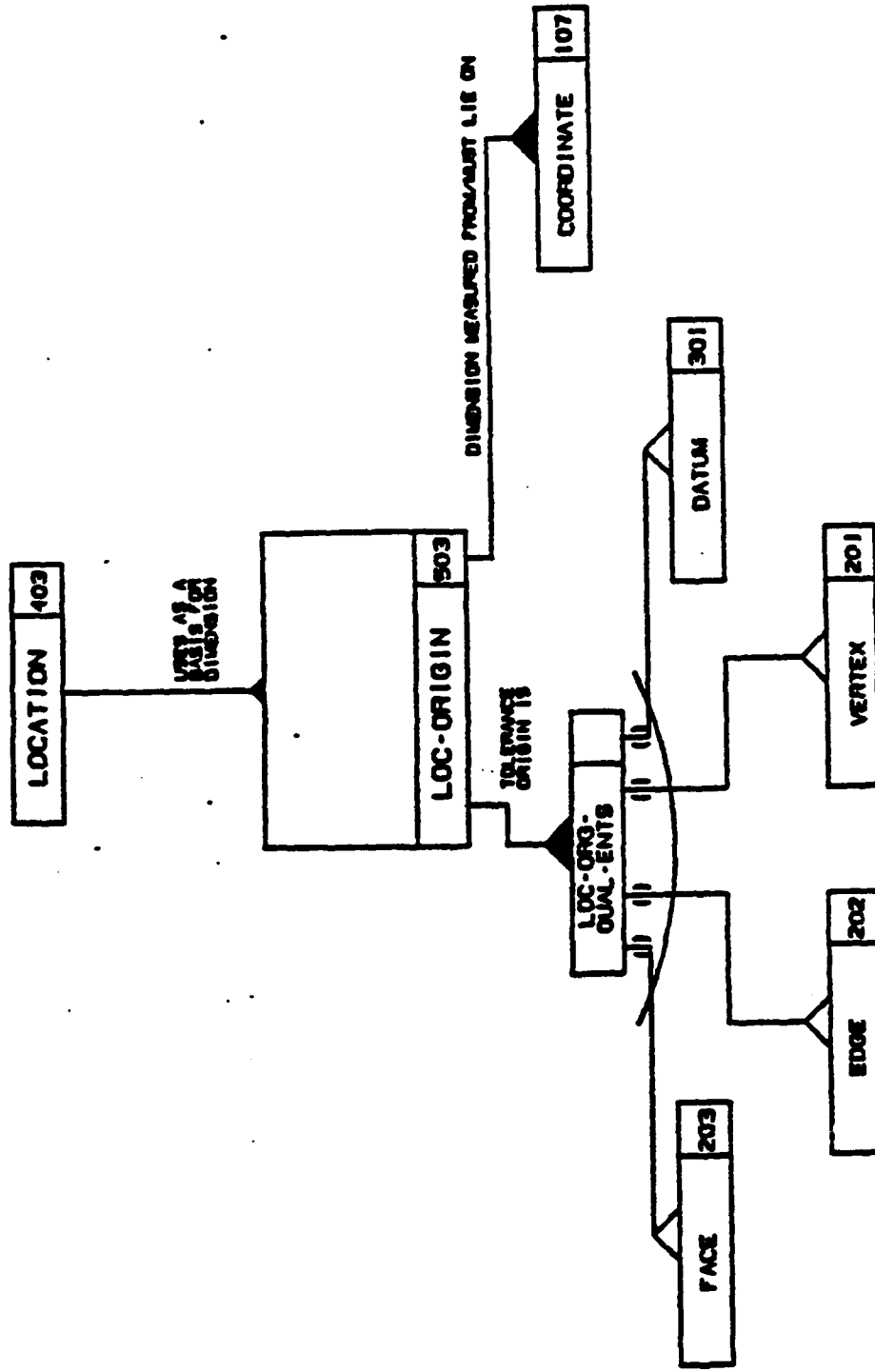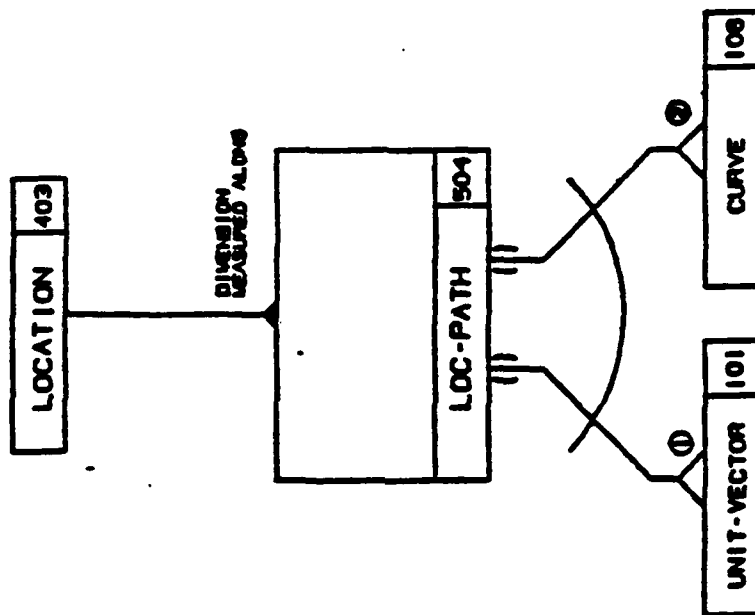
MAR 26 1986

LOCATION 103

LINE IS A
DIMENSION

LOC-ORIGIN 503

DIMENSION MEASURED FROM/MUST LIE ON

COORDINATE 107

LOC-ORG-QUAL-ENTS

DATUM 301

VERTEX 201

EDGE 202

FACE 203

LB-503-LOCATION ORIGIN

LOCATION | 403

DIMENSION
FEATURES ALONG

LOC-PATH | 504

CURVE | 108

UNIT-VECTOR | 101

L8-504-LOCATION PATH

MAR 26 1986
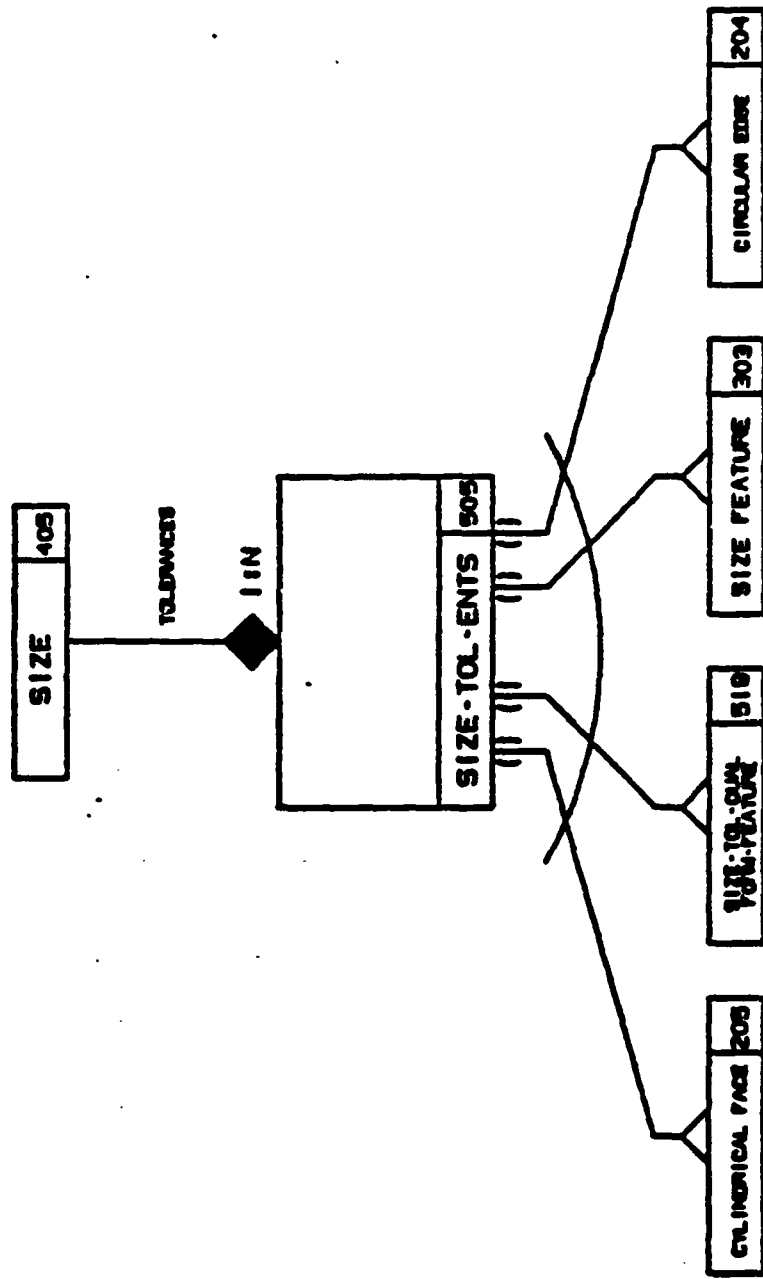
L11-505-SIZE TOLERANCED ENTITIES
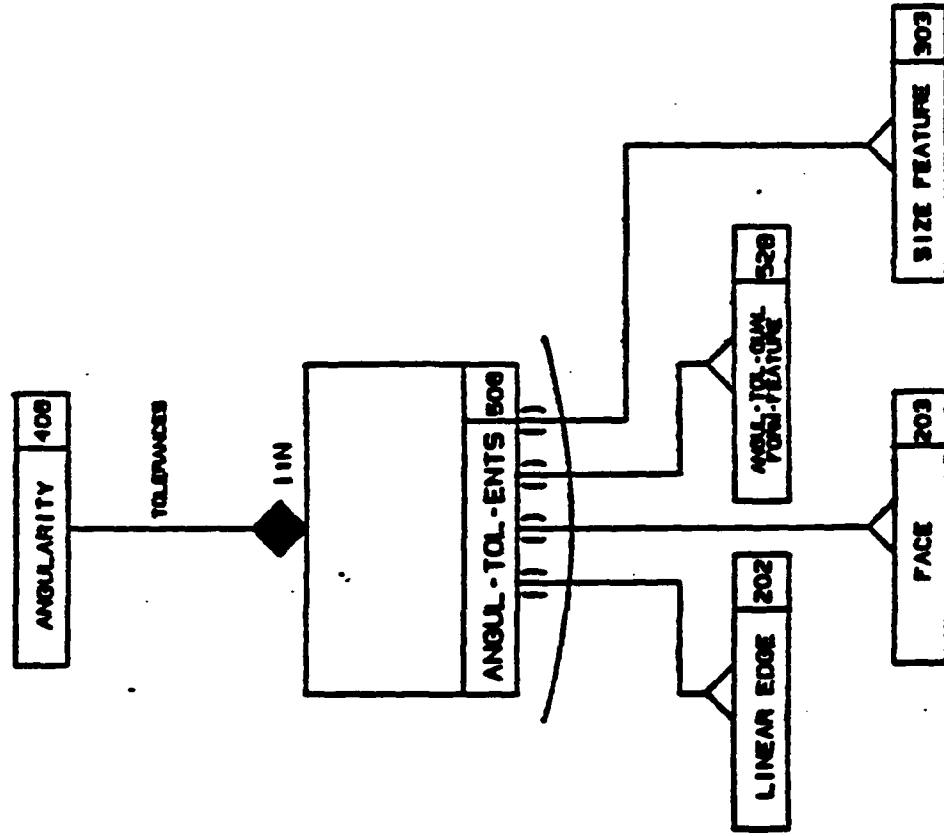
L30-500-ANGULARITY TOLERANCED ENTITIES
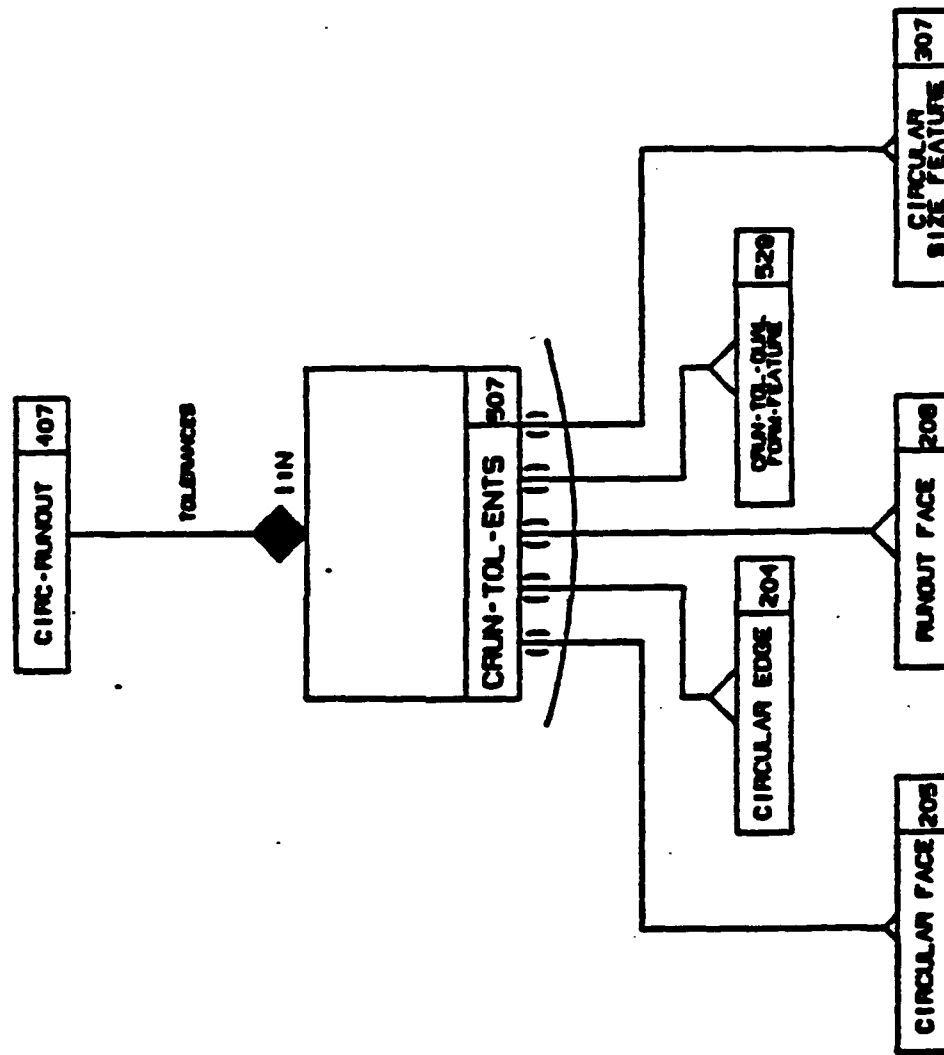
MAR 26 1986

MAR 26 1986

CIRC-RUNOUT | 407

TOLERANCES

1N

CRUN-TOL-ENTS | 507

CRUN-TOL-SIZE-FEATURE | 529

CIRCULAR SIZE FEATURE | 307

CIRCULAR EDGE | 204

RUNOUT FACE | 208

CIRCULAR FACE | 205

L40-507-CIRCULAR RUNOUT TOLERANCED ENTITIES

MAR 26 1986

L35-508-CIRCULARITY TOLERANCED ENTITIES

CONCENTRICITY 409

TOLERANCES

1 1 N

CONC-TOL-ENTS 509

CIRCULAR EDGE 204

CRE-1D-SUB-TOLM-FEATURE 521

CIRCULAR FACE 205

CIRCULAR SIZE FEATURE 307

L32-509-CONCENTRICITY TOLERANCED ENTITIES

MAR 26 1986

MAR 26 1986

CYLINDRICITY 410

TOLERANCES

I:N

CYL-TOL-ENTS 510

CYLINDRICAL
SIZE FEATURE 308

CYLINDRICAL-
TOLD FEATURE 525

CYLINDRICAL FACE 207
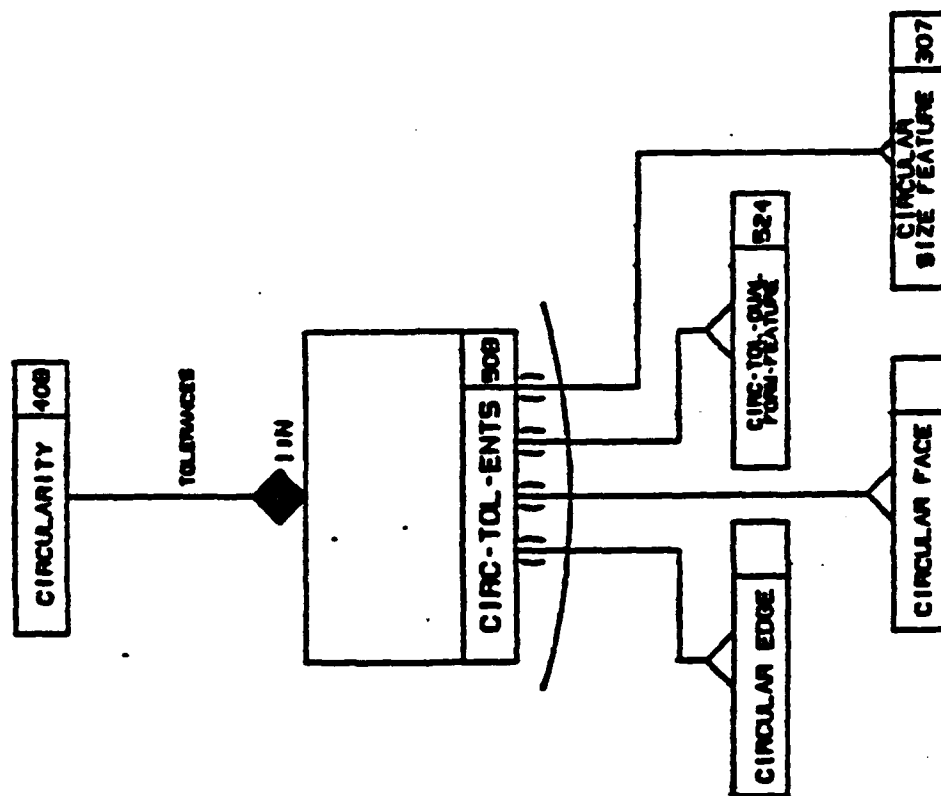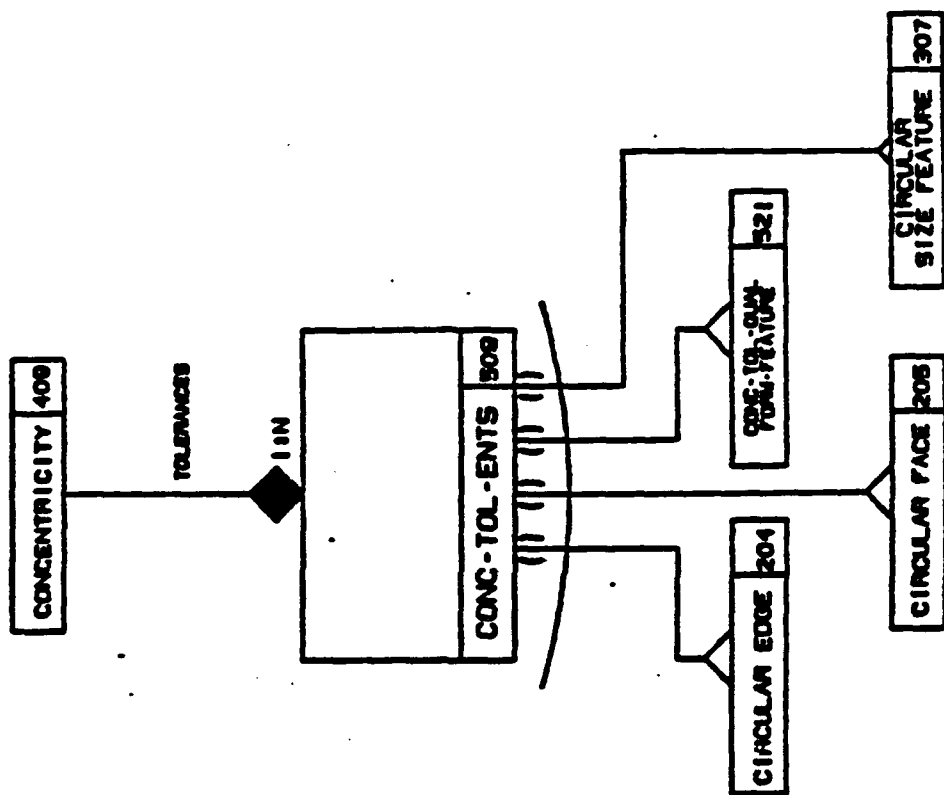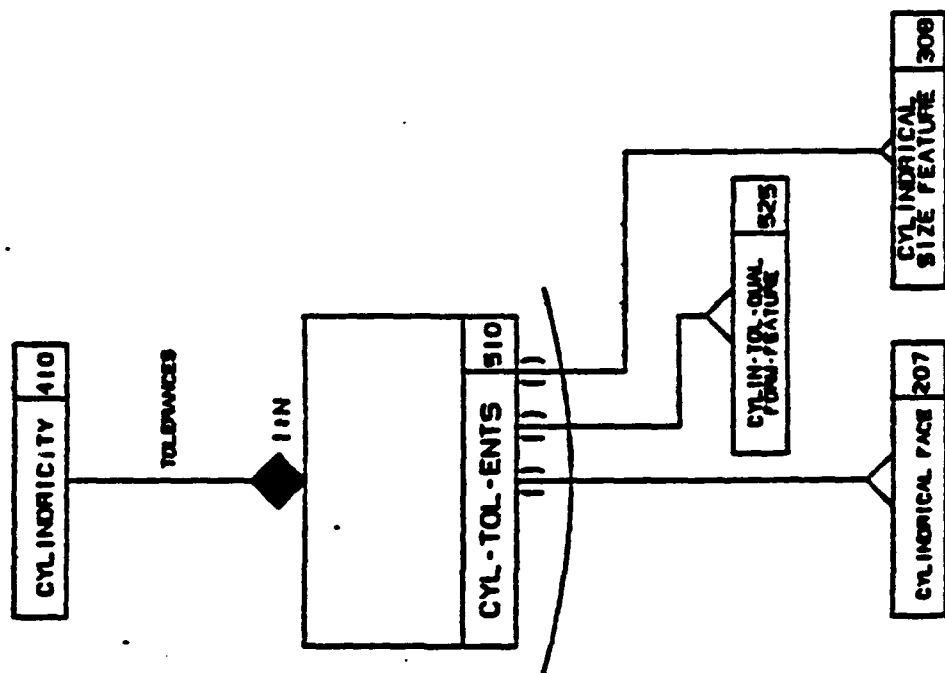
L38-510-CYLINDRICITY TOLERANCED ENTITIES
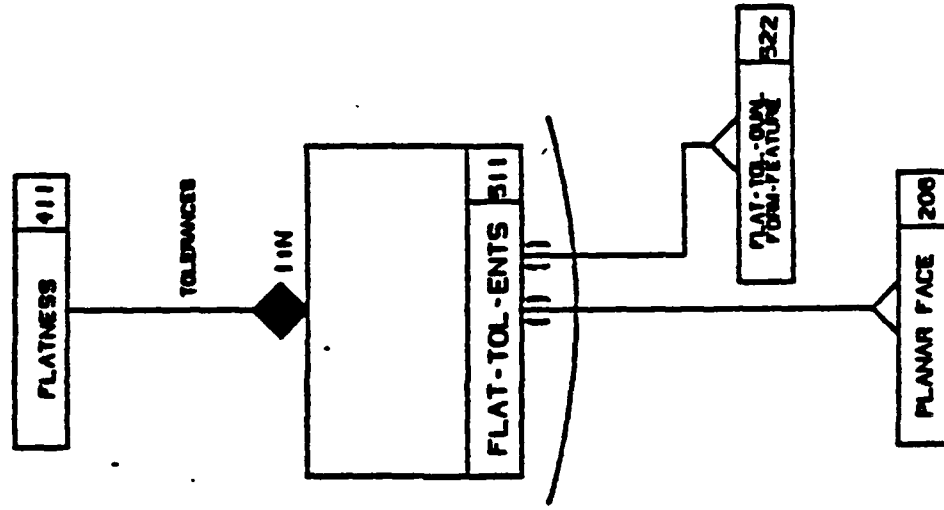
MAR 26 1986

L33-511-FLATNESS TOLERANCED ENTITIES
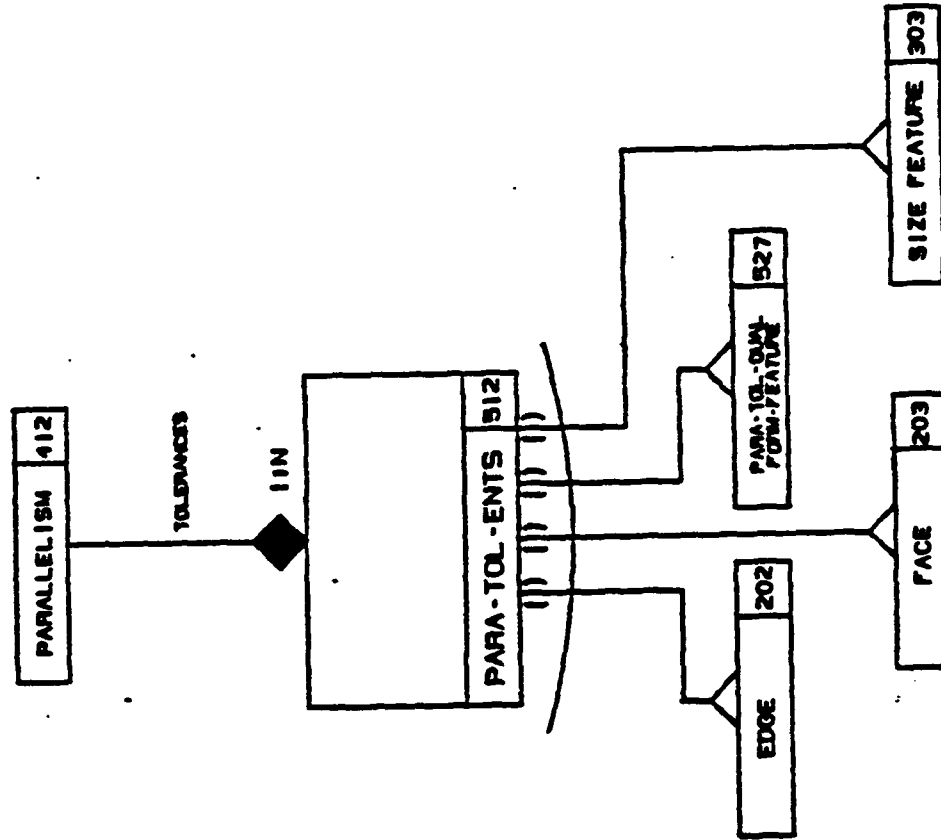
L38-512-PARALLELISM TOLERANCED ENTITIES
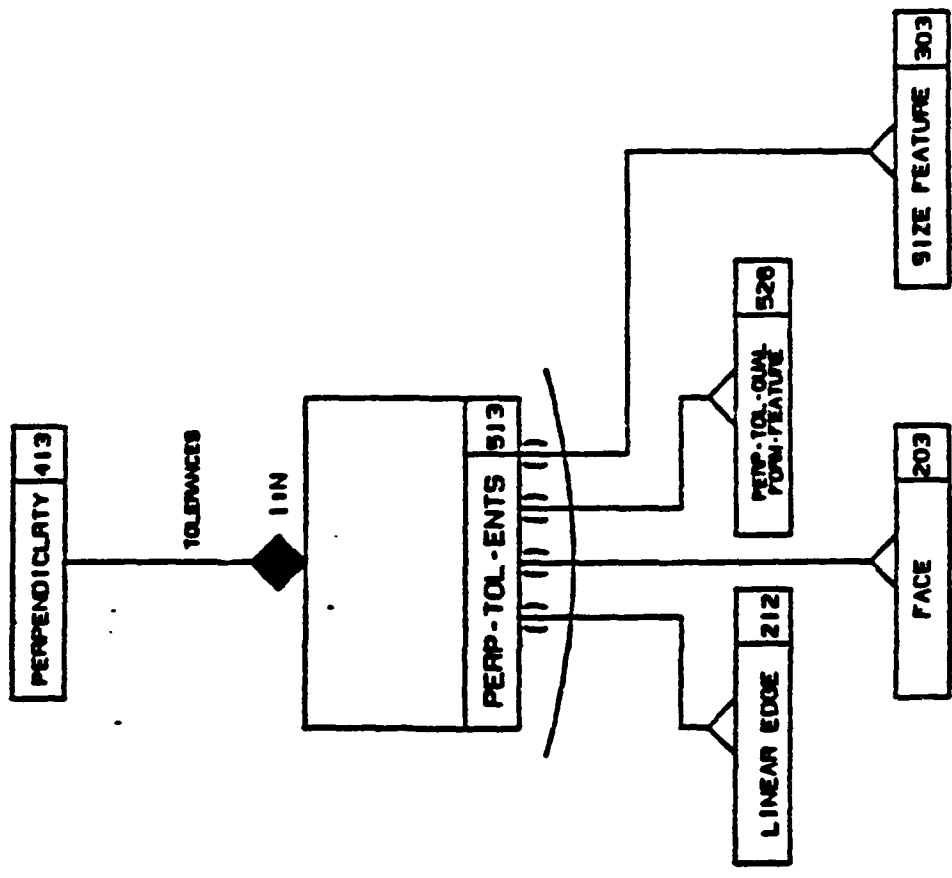
MAR 26 1986

L37-513-PERPENDICULRITY TOLERANCED ENTITIES
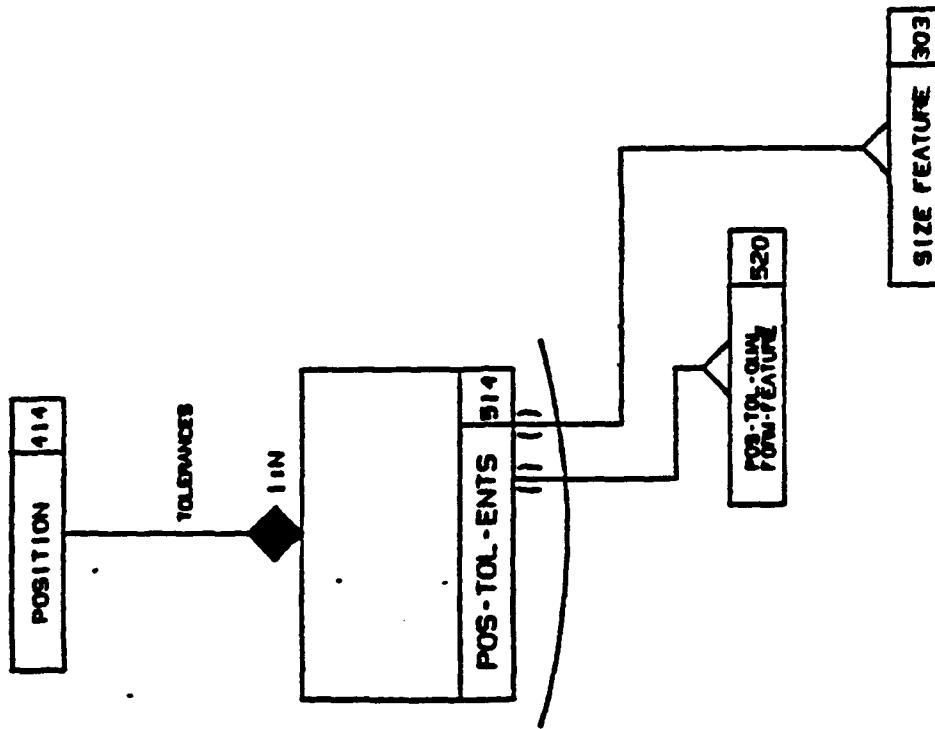
MAR 26 1986

L31-514-POSITION TOLERANCED ENTITIES
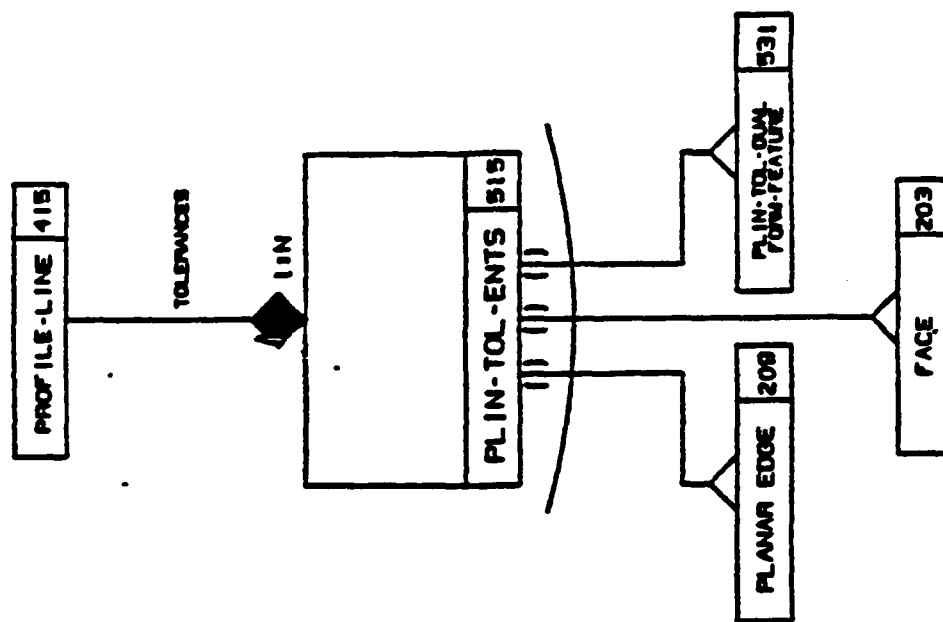
L43-515-PROFILE OF A LINE TOLERANCED ENTITIES
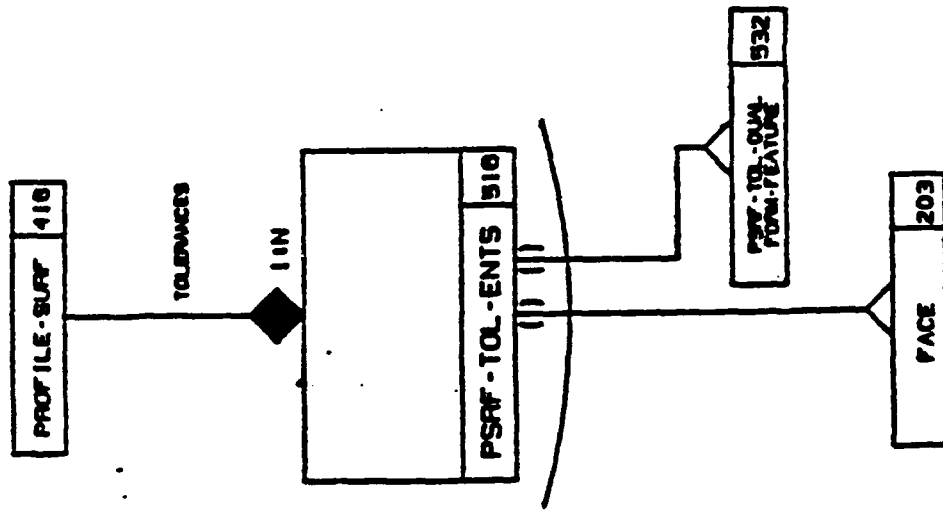
MAR 26 1976

L42-510-PROFILE OF A SURFACE TOLERANCED ENTITIES
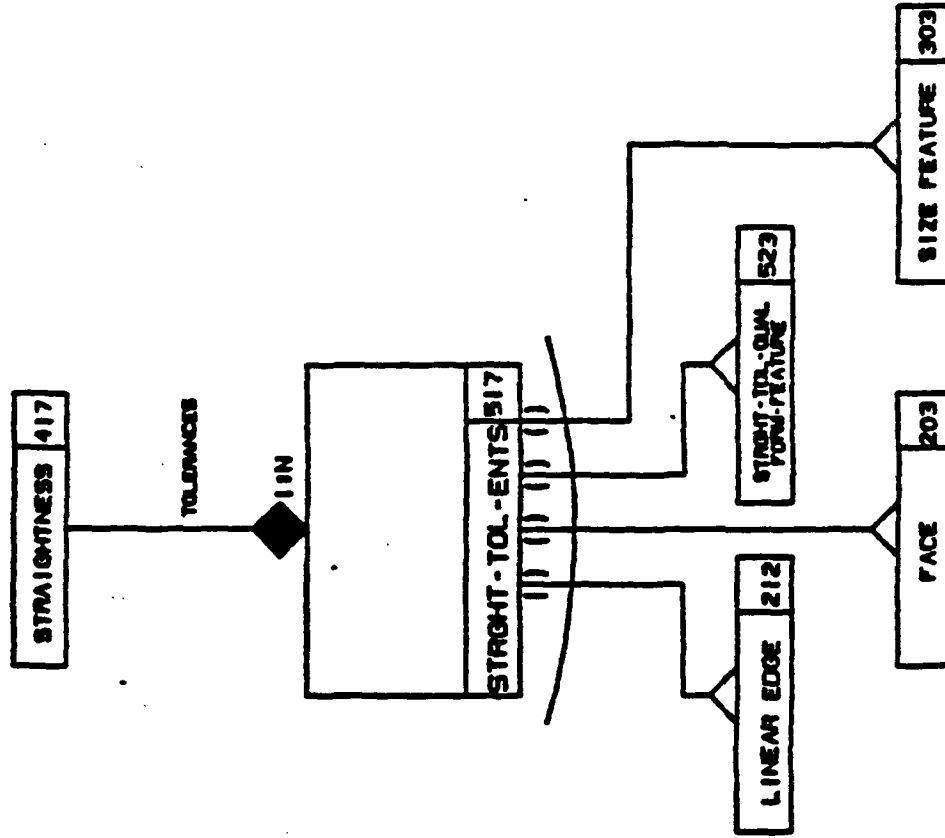
MAR 26 1986

L34-517-STRAIGHTNESS TOLERANCED ENTITIES
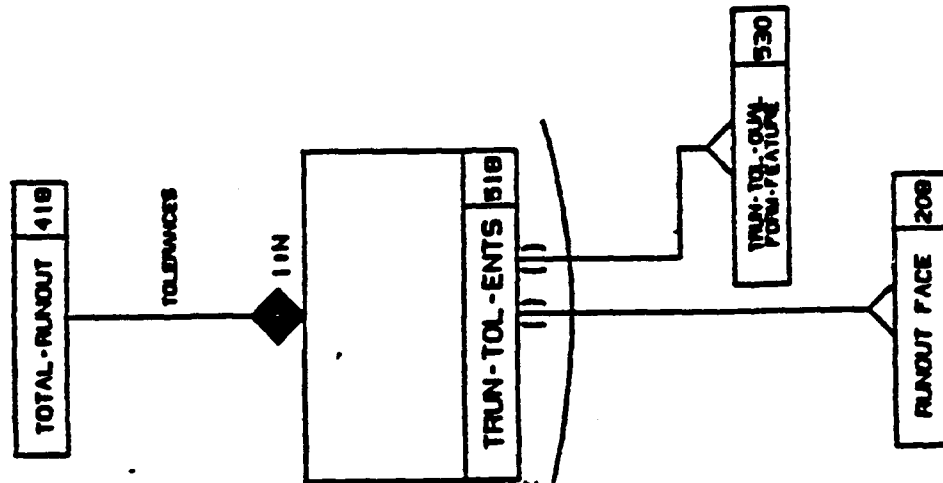
TOTAL-RUNOUT | 419

TOLERANCES

1:N

TRUN-TOL-ENTS | 518

TRUN-TB-RUN-FCRM-FEATURE | 530

RUNOUT FACE | 209

THE APPLICATION OF NON-DIRECTIONAL TOTAL RUNOUT TOLERANCE TO NON-CYLINDRICAL FACES IS NOT HERE SPECIFIED TO ANGLE OR NED. IT IS ASSUMED THAT THIS IS DUE TO PROHIBITION OF MECHANICAL LIMITATION ON MEASURING THE TOTAL RUNOUT OF THOSE SURFACES. IN PARTICULAR, FACES DEFINED BY CONICAL AND TOROIDAL SURFACES.

L41-518-TOTAL RUNOUT TOLERANCED ENTITIES

Appendix C5.2

Task 2

<u>Tolerancing Qualified Model (NIAM)</u>

(DISCIPLINE)

24
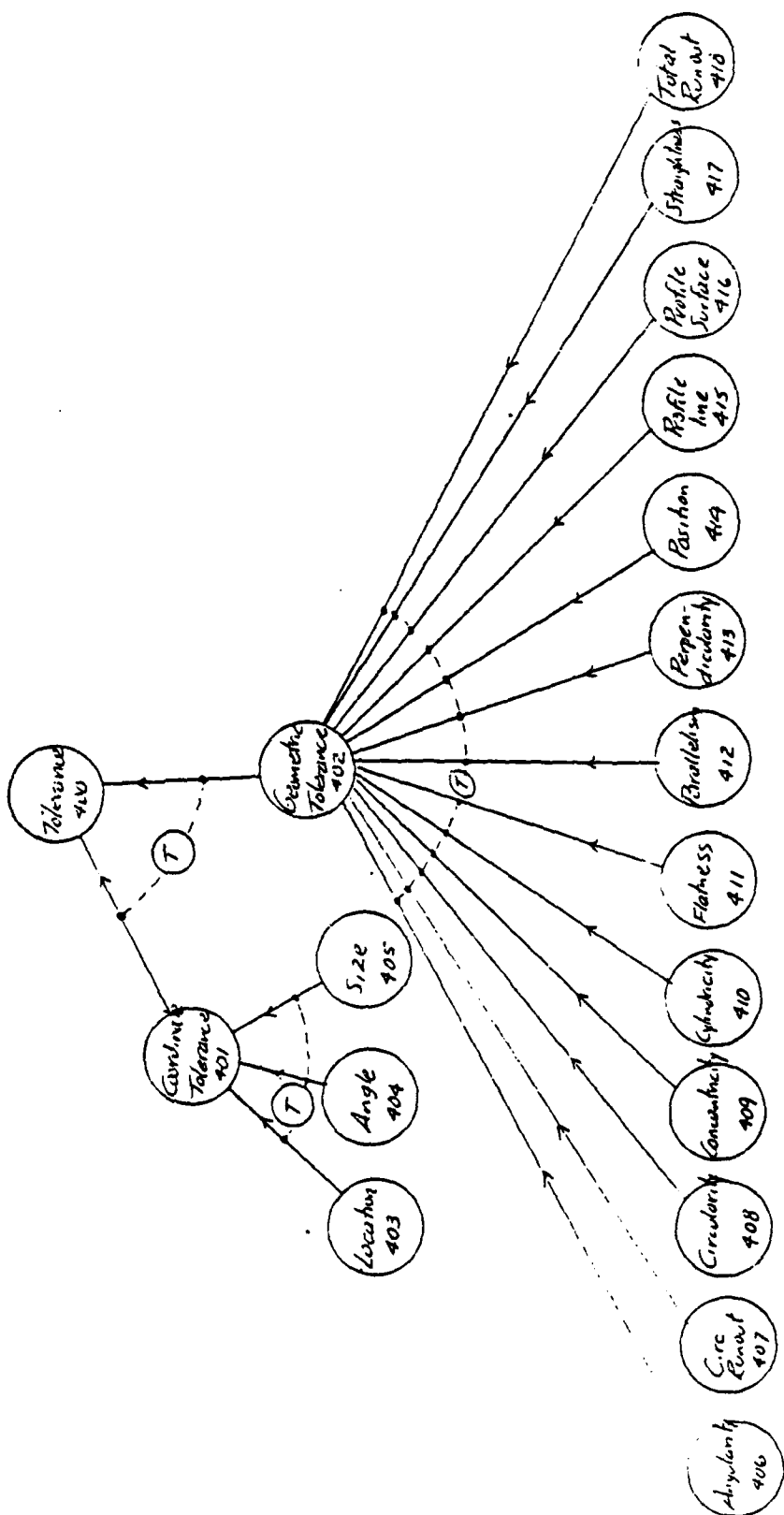
400 TOLERANCE

QUALIFIED

QUALIFIED MODEL

403   LOCATION TOLERANCE
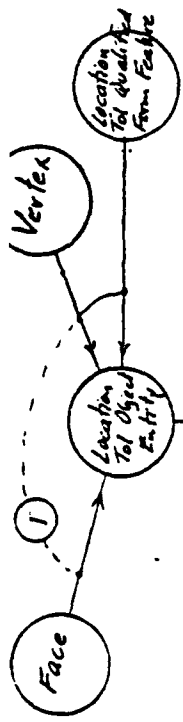
404 ANGLE TOLERANCE
QUALIFIED
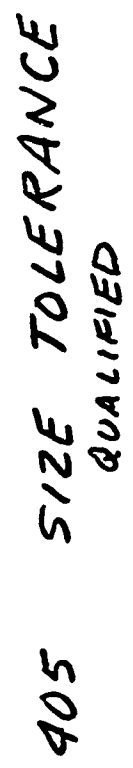
405   SIZE TOLERANCE
QUALIFIED

406   ANGULARITY TOLERANCE

QUALIFIED

Tol Span

Point Vector

Angularity Tol 406

Angularity Tol Datum EATN

Size Feature

Angularity Tol Qualified Form Feature

Angularity Tol Object EATN

Face

Edge

has

U

is primary reference for

is secondary reference for

is tertiary reference for

specifies projected tol zone for

has projected tol zone given by

has

applies to

T
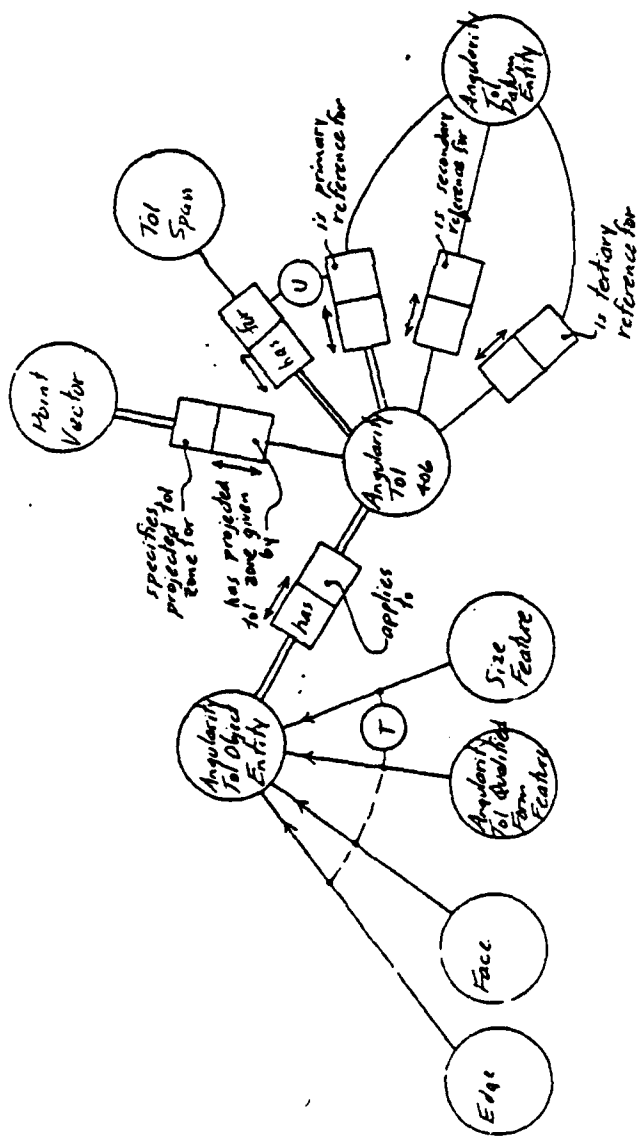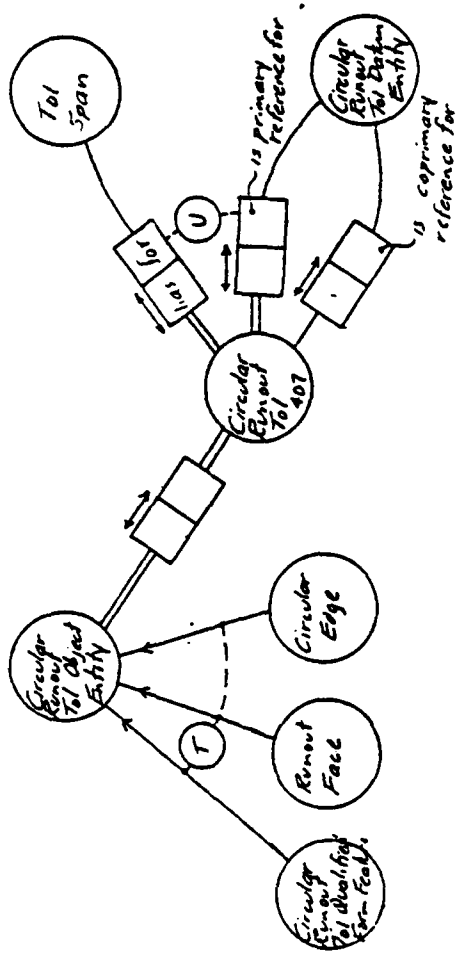
407 CIRCULAR RUNOUT TOLERANCE
QUALIFIED
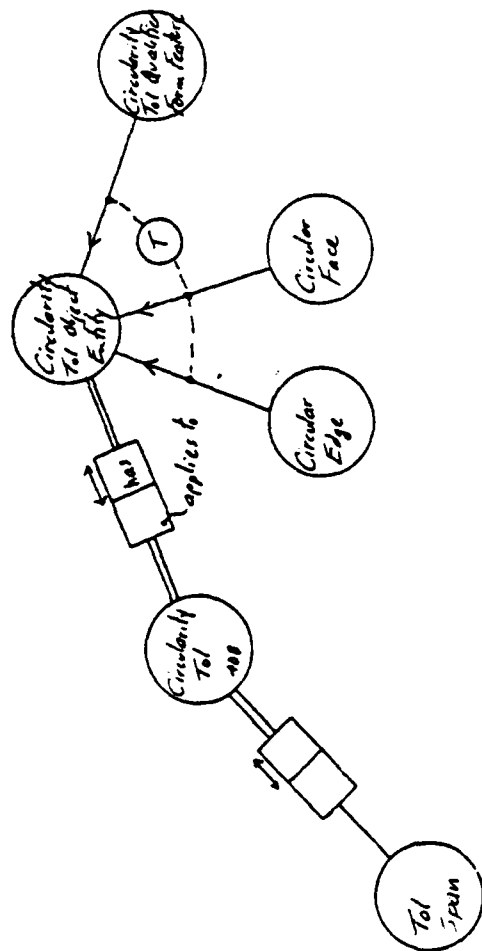
408   CIRCULARITY TOLERANCE
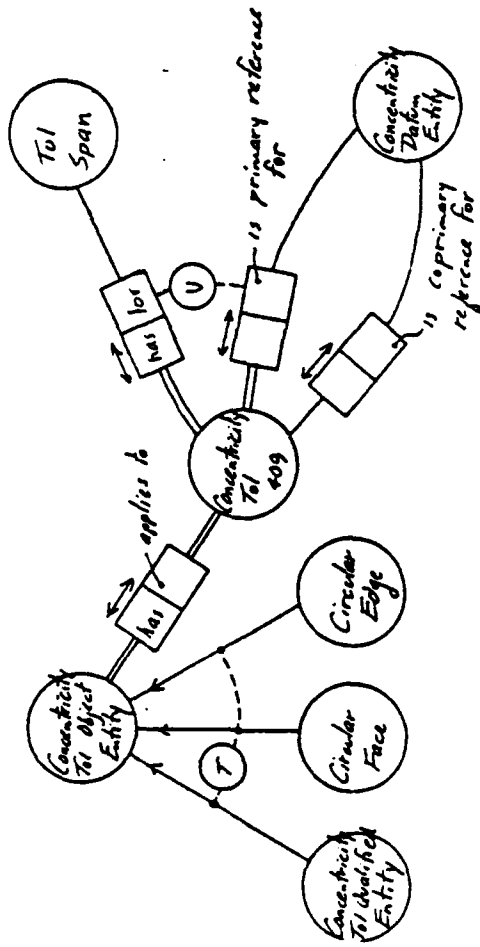
QUALIFIED

409 CONCENTRICITY TOLERANCE
QUALIFIED
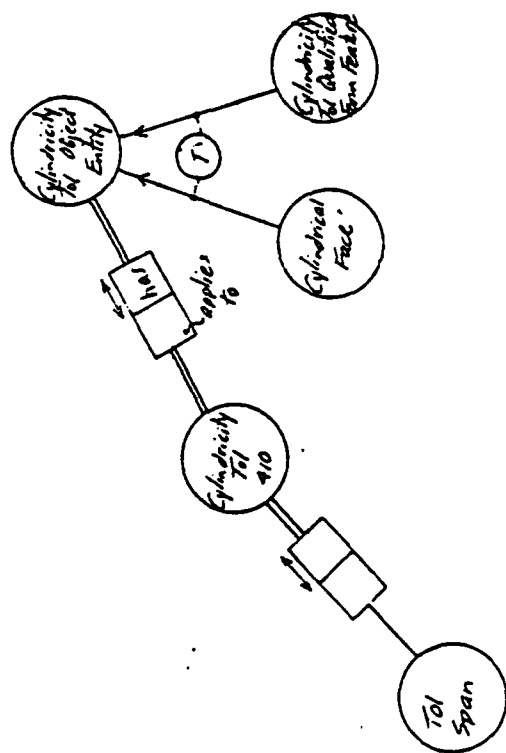
410   CYLINDRICITY TOLERANCE
QUALIFIED

411 FLATNESS TOLERANCE
QUALIFIED

5-1-01
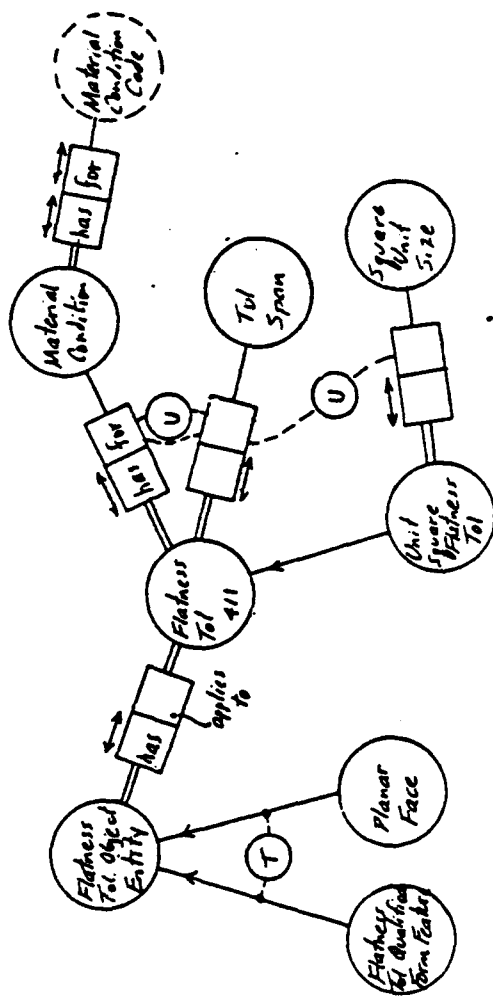
412 PARALLELISM TOLERANCE QUALIFIED
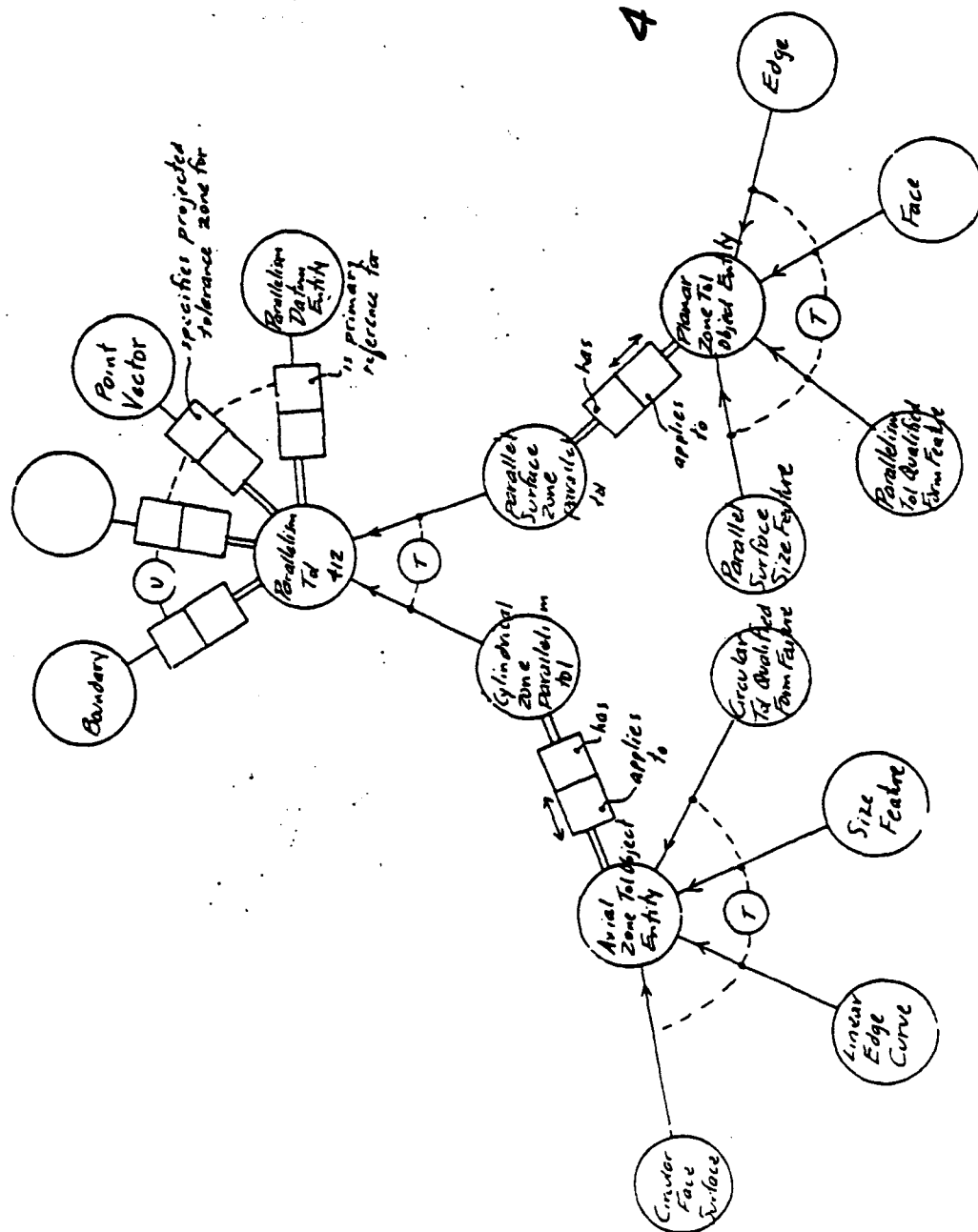
PERPENDICULARITY
TOLERANCE
QUALIFIED

413
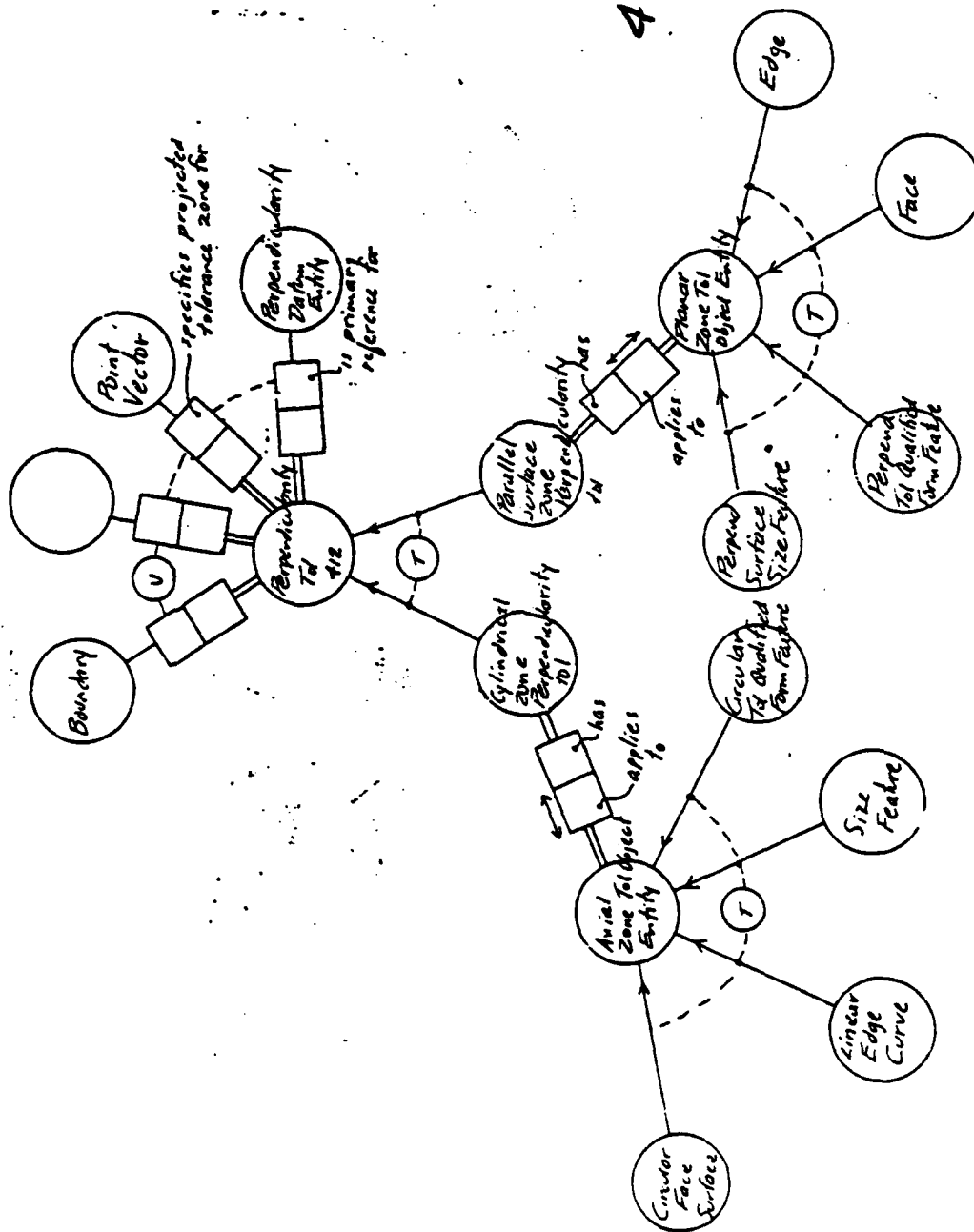
414 POSITION TOLERANCE
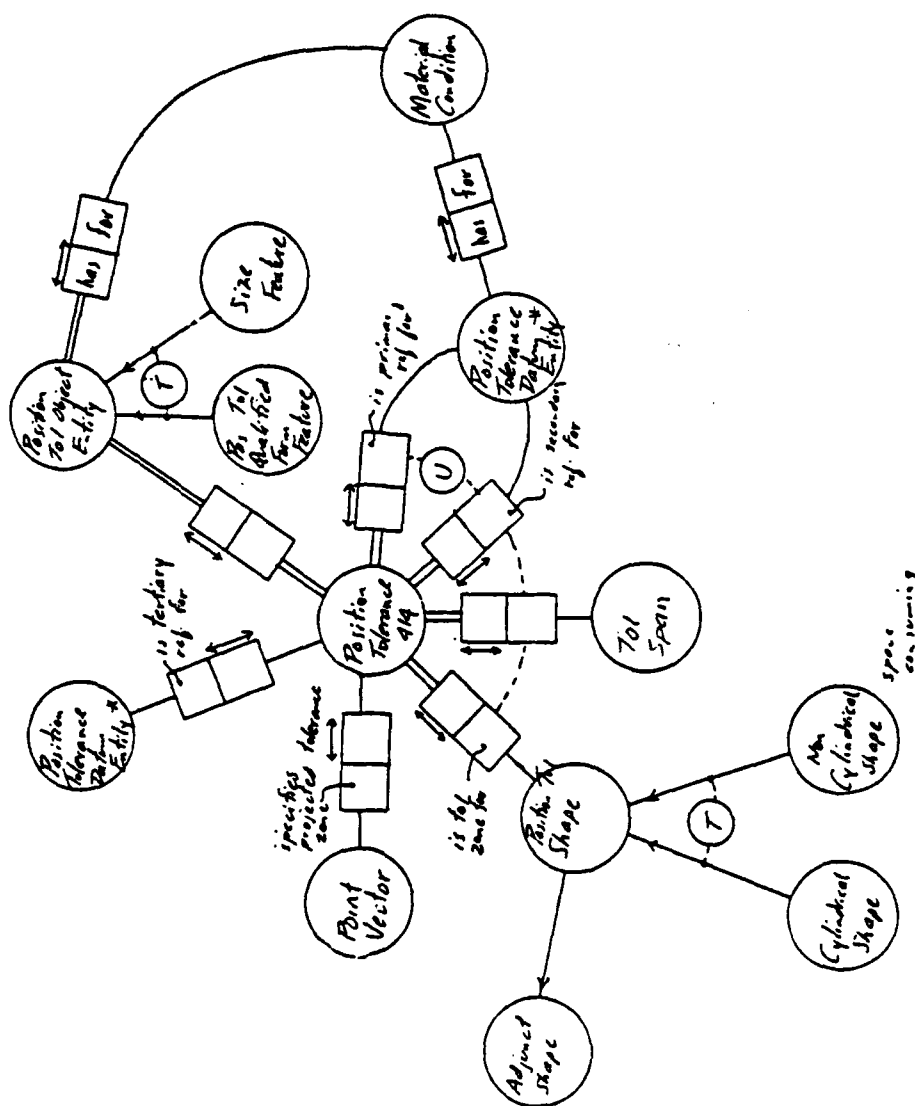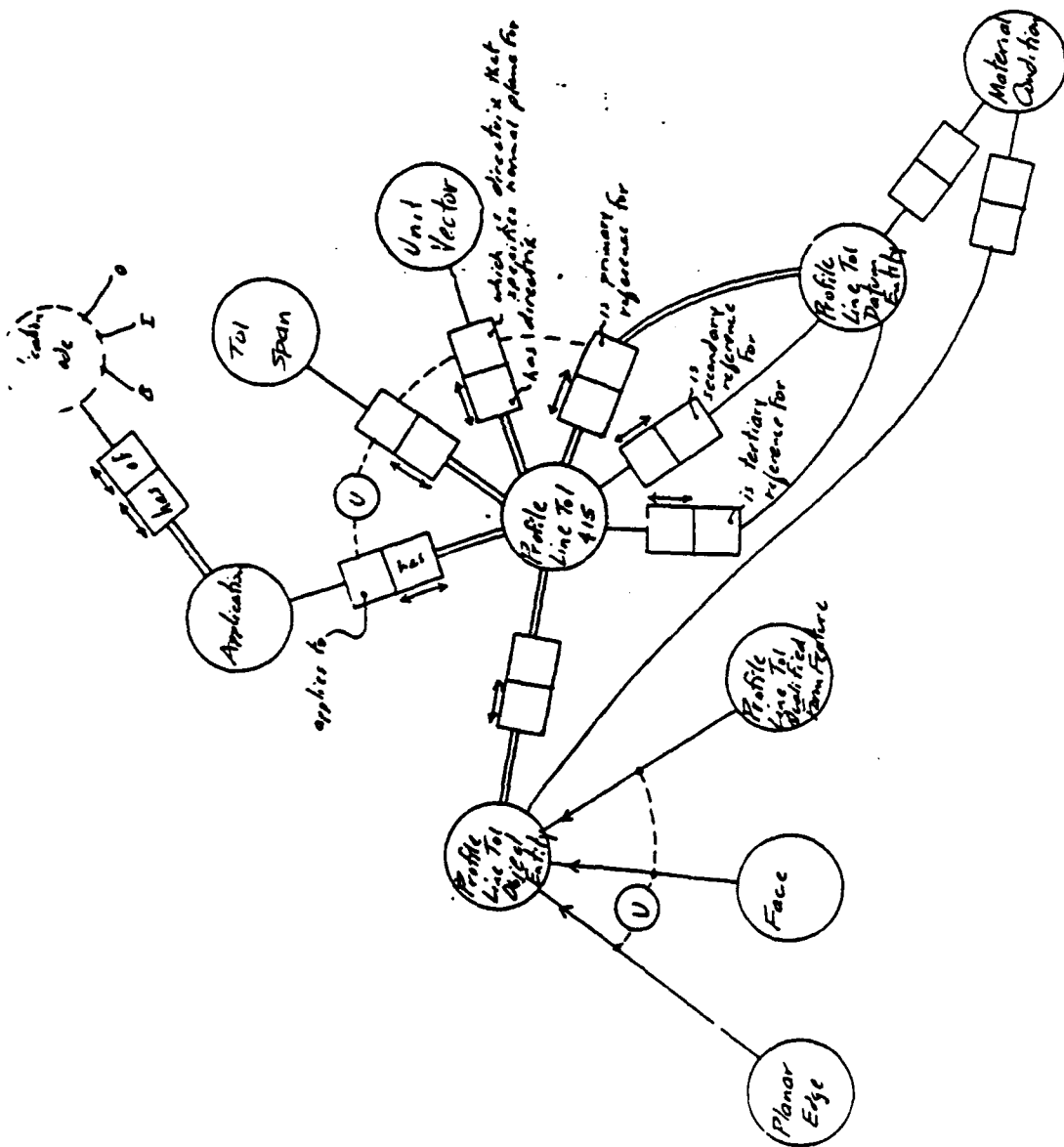
QUALIFIED

415 PROFILE LINE TOLERANCE QUALIFIED
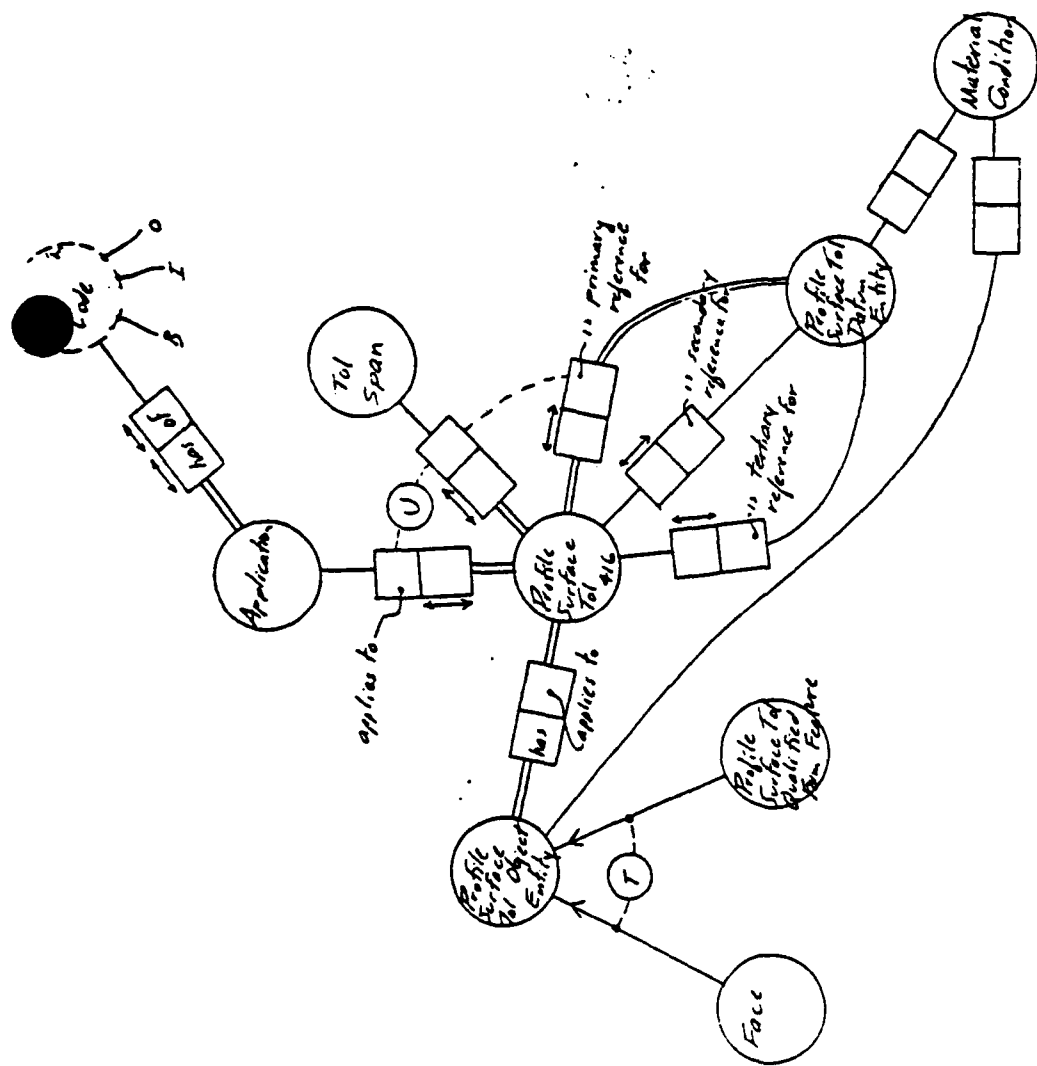
416 PROFILE SURFACE TOLERANCE

417 Straightness Tolerance
Qualified

Material Condition Code

has of

Material Condition

to roy

Unit Vector

Tol Span

Straightness Tolerance 417

Surface Element Straightness

Surface Element Straightness Tol Object Entity

Face

Edge

T

Magnitude

is unit length for

Feature Center Surface Straightness

Unit Vector

Feature axis Straightness

Feature Axis Straightness Tol Object Entity

Straightness Tol Qualified Form Feature

Size Feature

Feature Center Surface Straightness Object Entity

Parallel Surface Straightness Size Feature

Parallel Surface Tol Unqualified Form Feature

T

418 TOTAL RUNOUT TOLERANCE
QUALIFIED

Appendix C5.3

Task 2

<u>Tolerancing Global Model (NIAM)</u>

(DISCIPLINE)
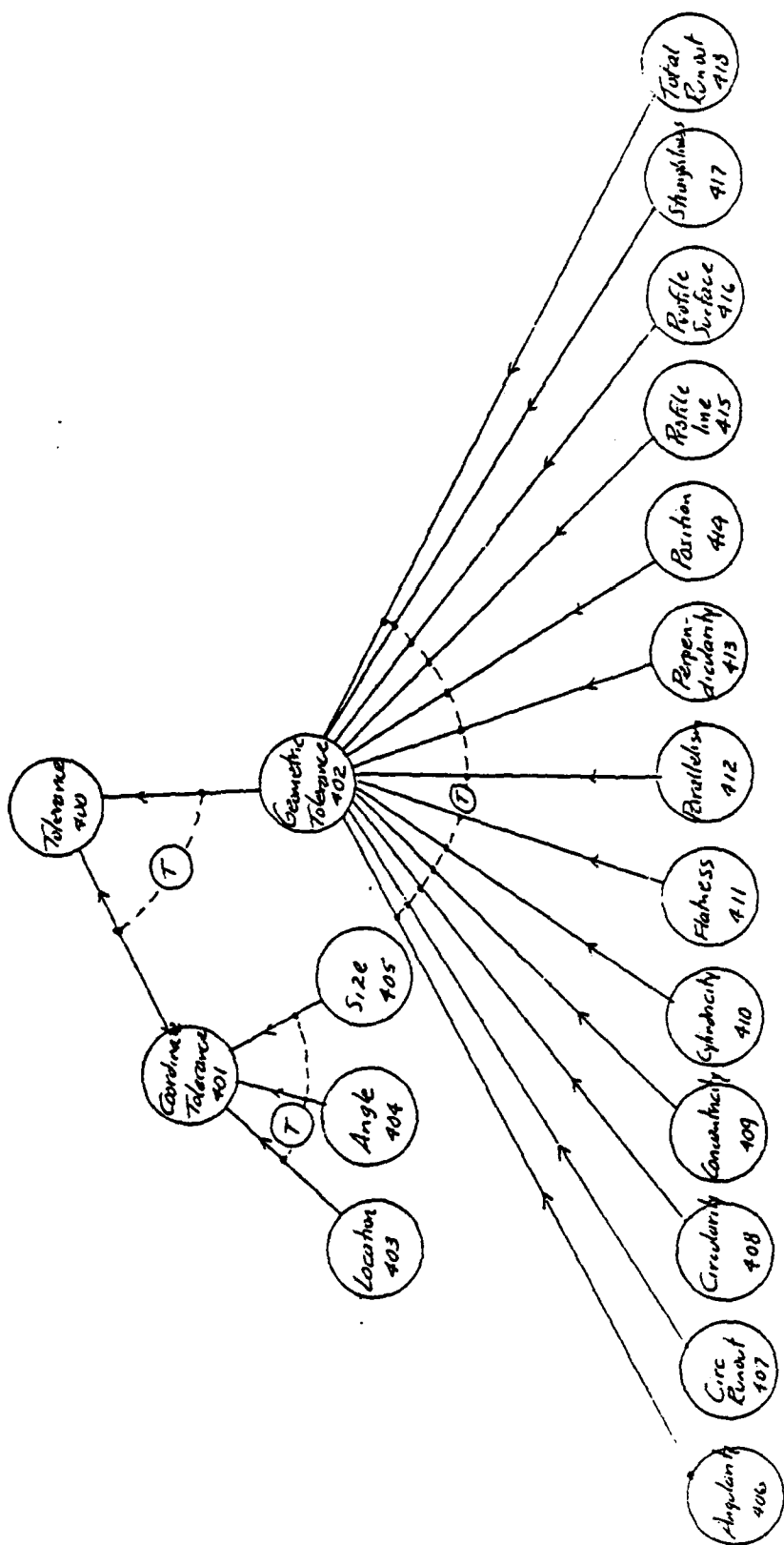
400 TOLERANCE

GLOBAL MODEL

GLOBAL MODEL

403   LOCATION TOLERANCE

404 ANGLE TOLERANCE
GLOBAL MODEL
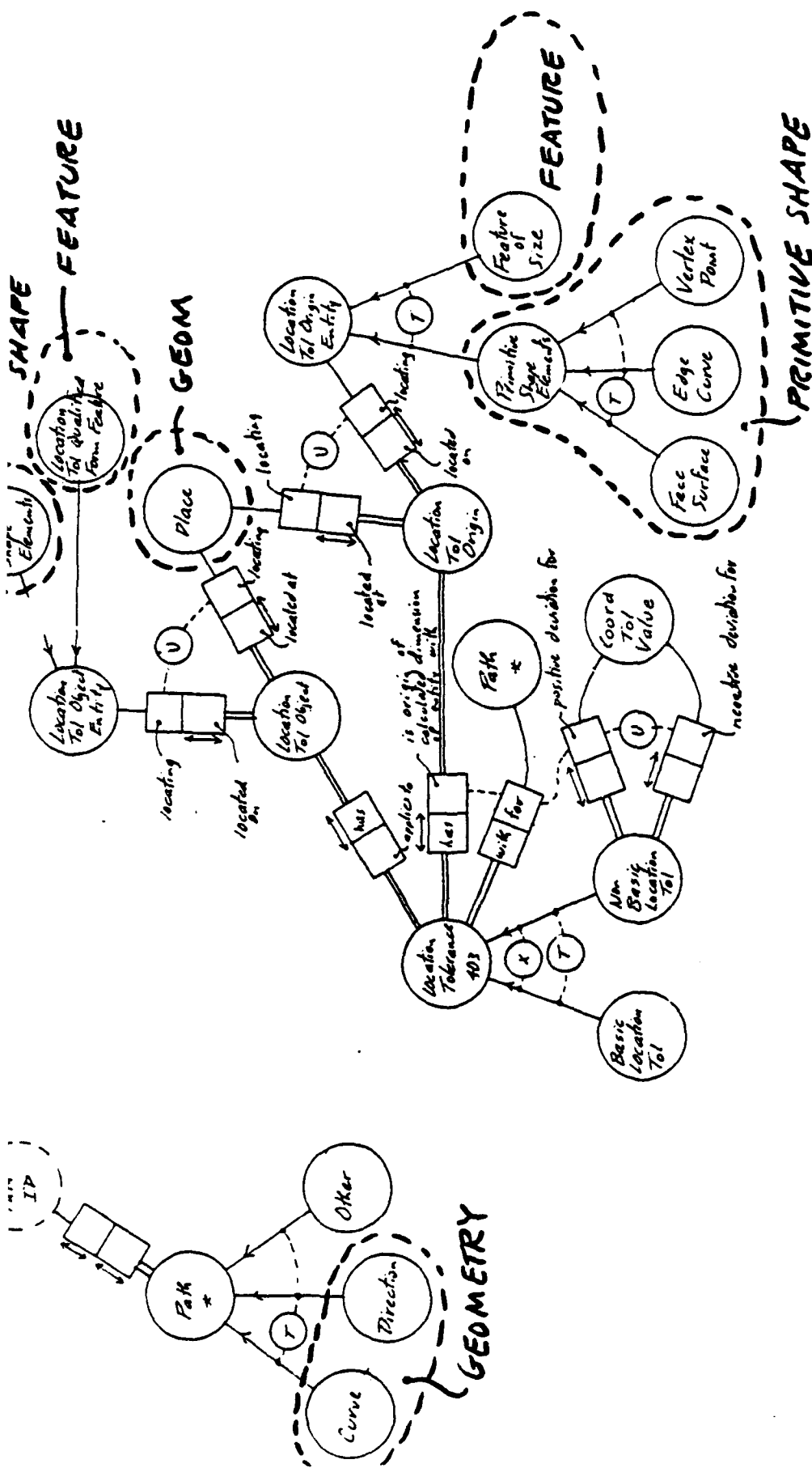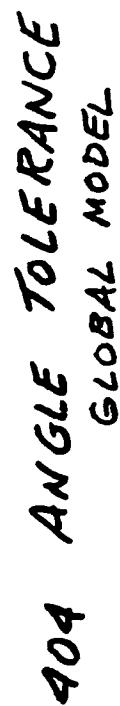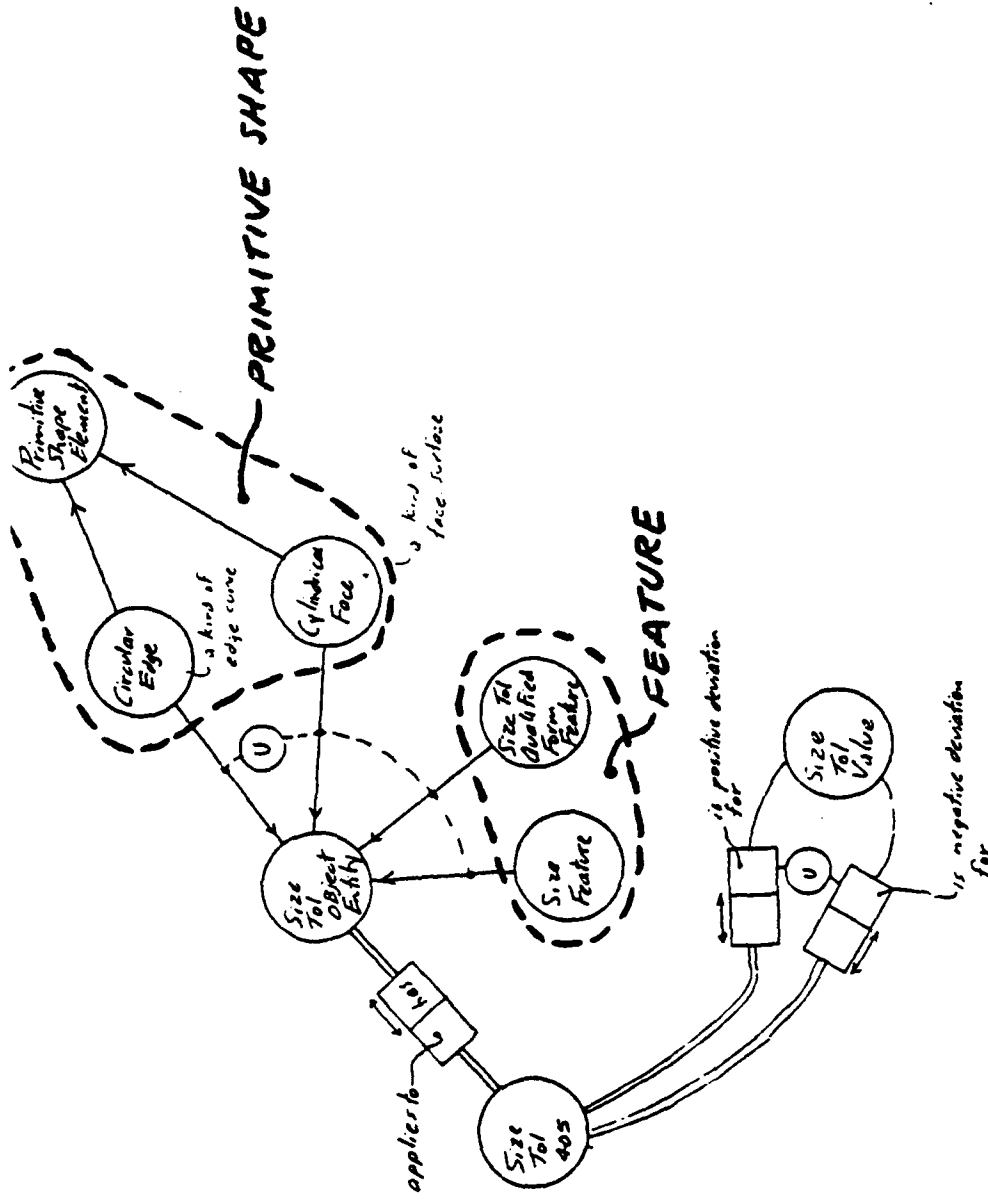
405   SIZE TOLERANCE
GLOBAL MODEL

406   ANGULARITY TOLERANCE
GLOBAL MODEL

GEOMETRY

407 CIRCULAR RUNOUT TOLERANCE
GLOBAL MODEL

PRIMITIVE SHAPE

FEATURE

FEATURE

PRIMITIVE SHAPE

GEOMETRY

408   CIRCULARITY   TOLERANCE
GLOBAL MODEL

GEOMETRY

Magnitude

is tolerance span for

is primary reference for

Concentricity Datum Entity

is coprimary reference for

U

Concentricity Tol 409

applies to

has

Circular Edge

Circular Face

PRIMITIVE SHAPE

Concentricity Tol Object Entity

T

Concentricity Tol Identified Entity

FEATURE

409   CONCENTRICITY TOLERANCE
GLOBAL MODEL

5-1-80

FEATURE

PRIMITIVE SHAPE

GEOMETRY

Cylindricity Tol Quality From Feature

Cylindrical Face

Cylindricity Tol Object Entry

has

applies to

Cylindricity Tol 410

is tol span for

Magnitude

410   CYLINDRICITY TOLERANCE
GLOBAL MODEL
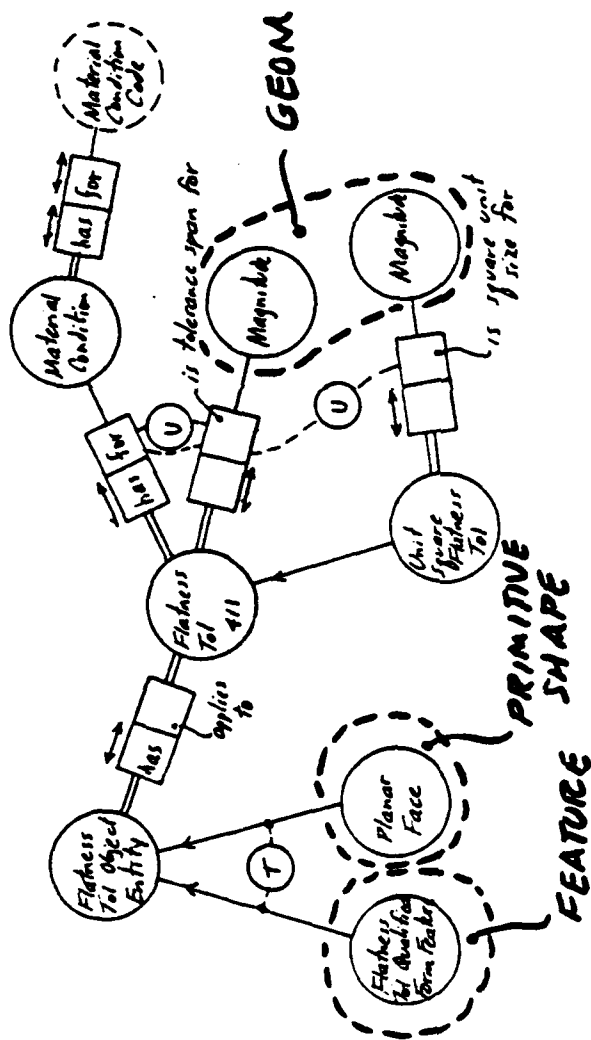
411 FLATNESS TOLERANCE
GLOBAL MODEL
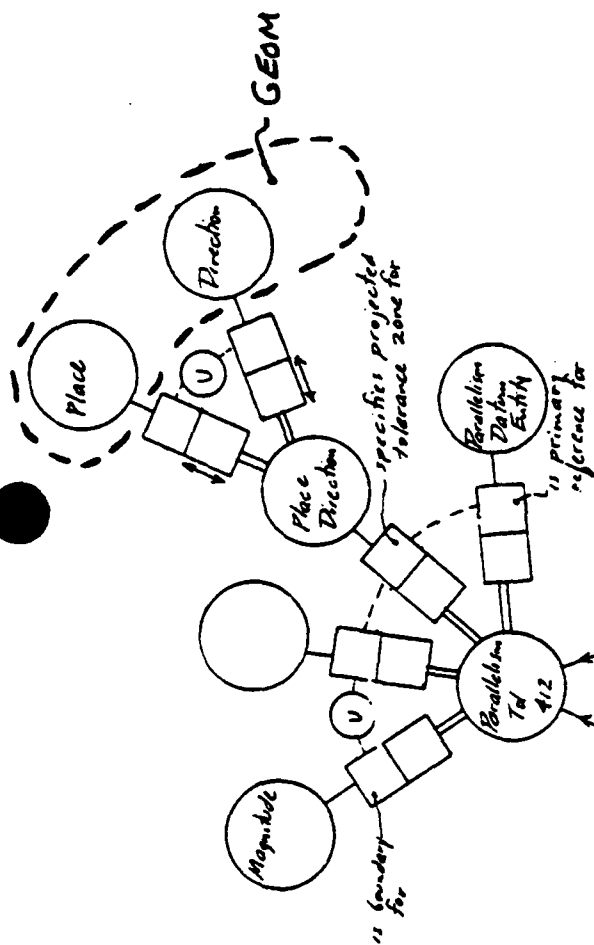
412 PARALLELISM
TOLERANCE

GLOBAL MODEL
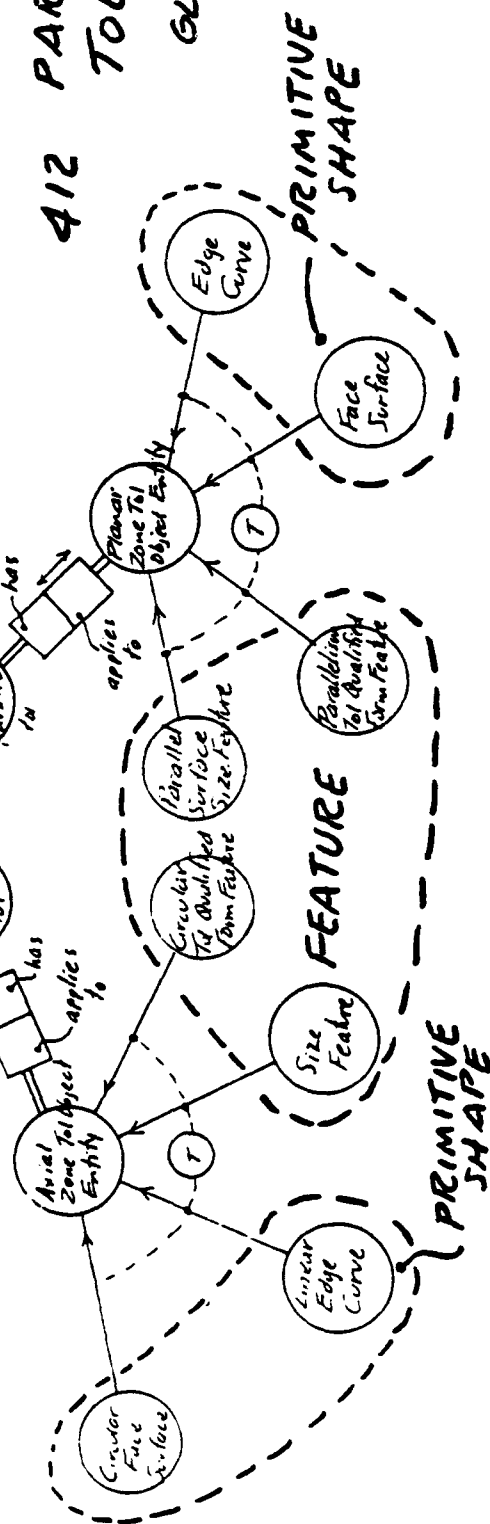
PERPENDICULARITY
TOLERANCE
GLOBAL MODEL

413
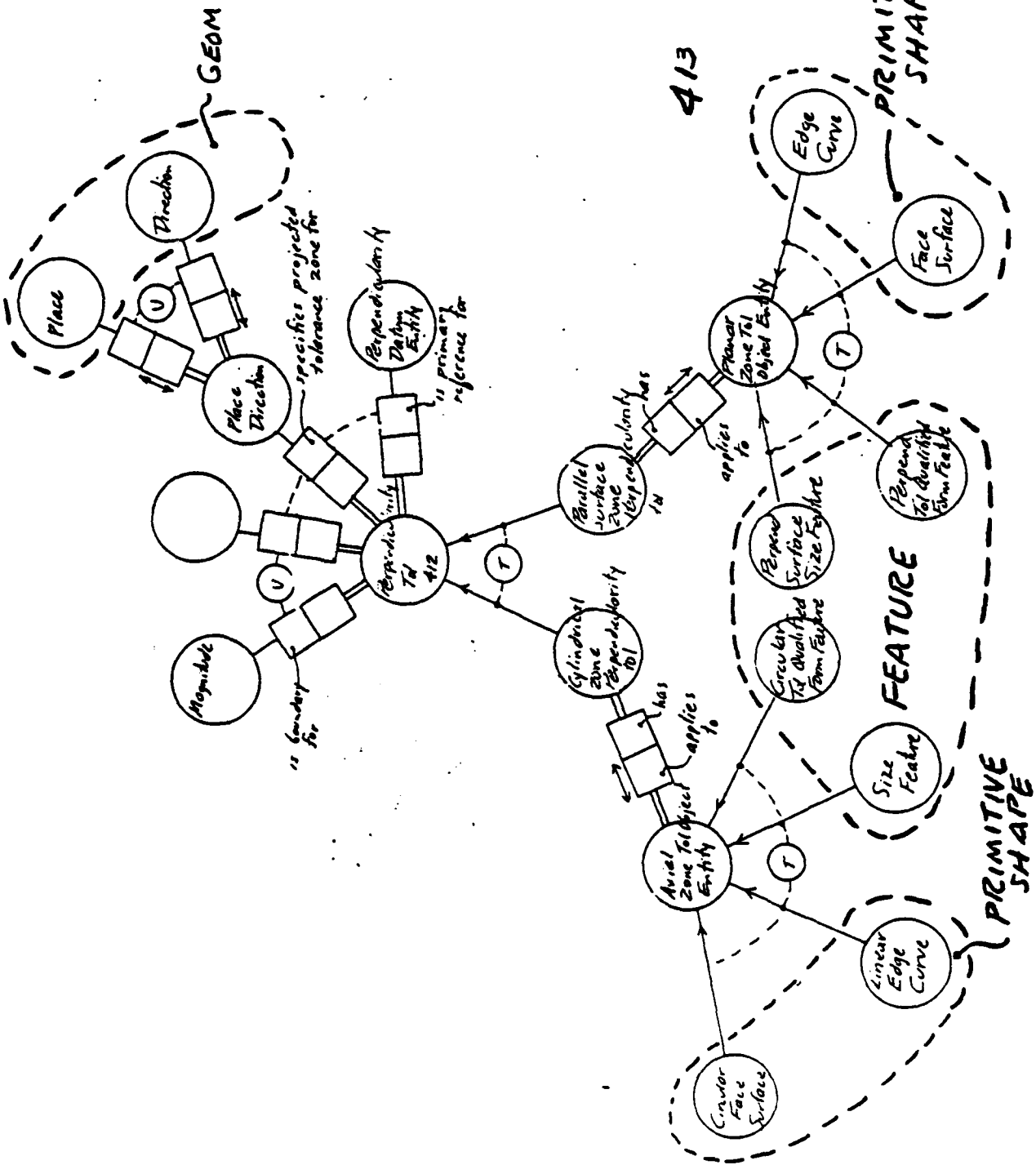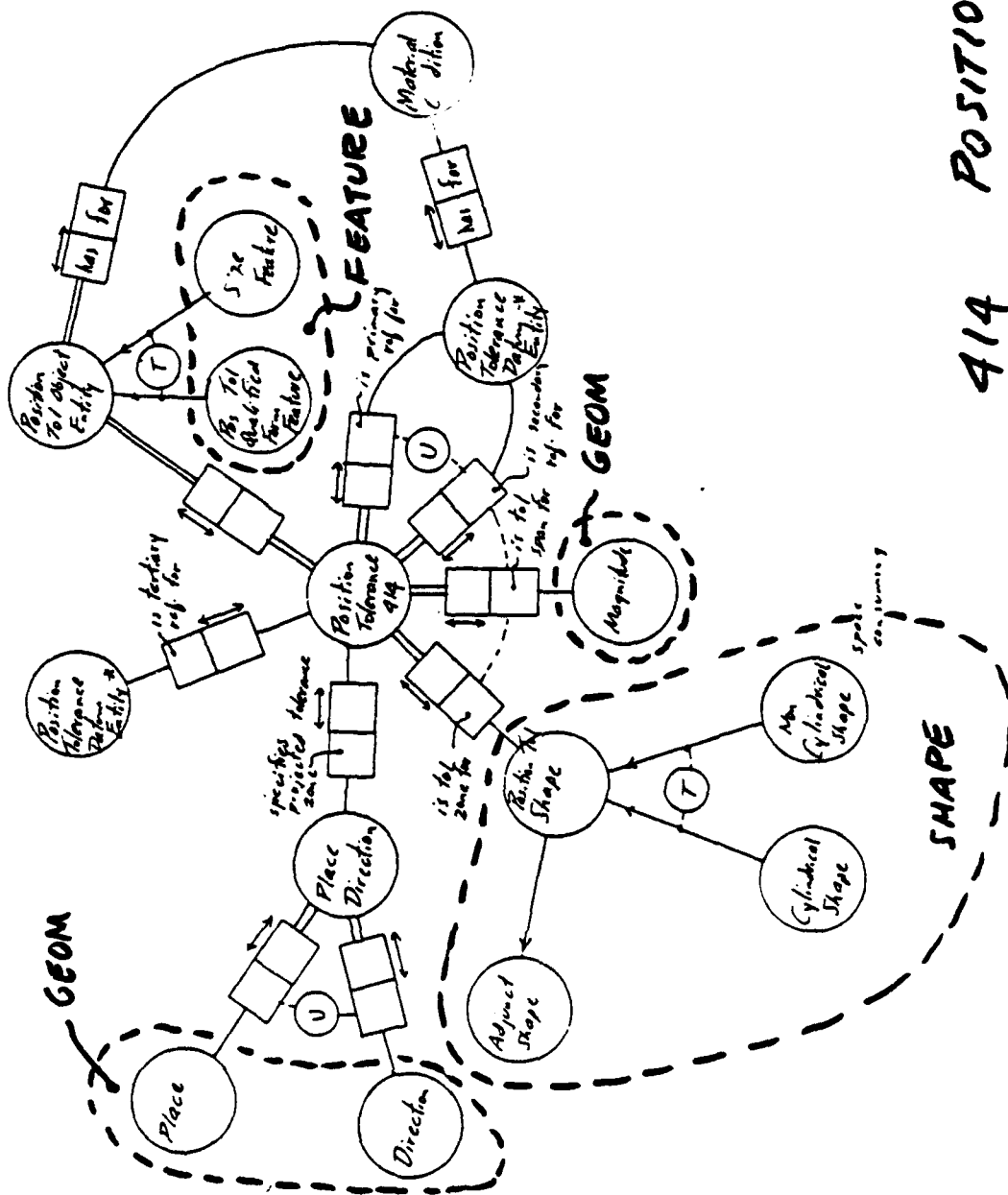
POSITION TOLERANCE
GLOBAL MODEL

414

415   PROFILE LINE TOLERANCE
GLOBAL MODEL

GEOMETRY

FEATURE

PRIMITIVE SHAPE

416 PROFILE SURFACE TOLERANCE
GLOBAL MODEL

417 Straightness Tolerance
GLOBAL MODEL

GEOMETRY

FEATURE

PRIMITIVE SHAPE
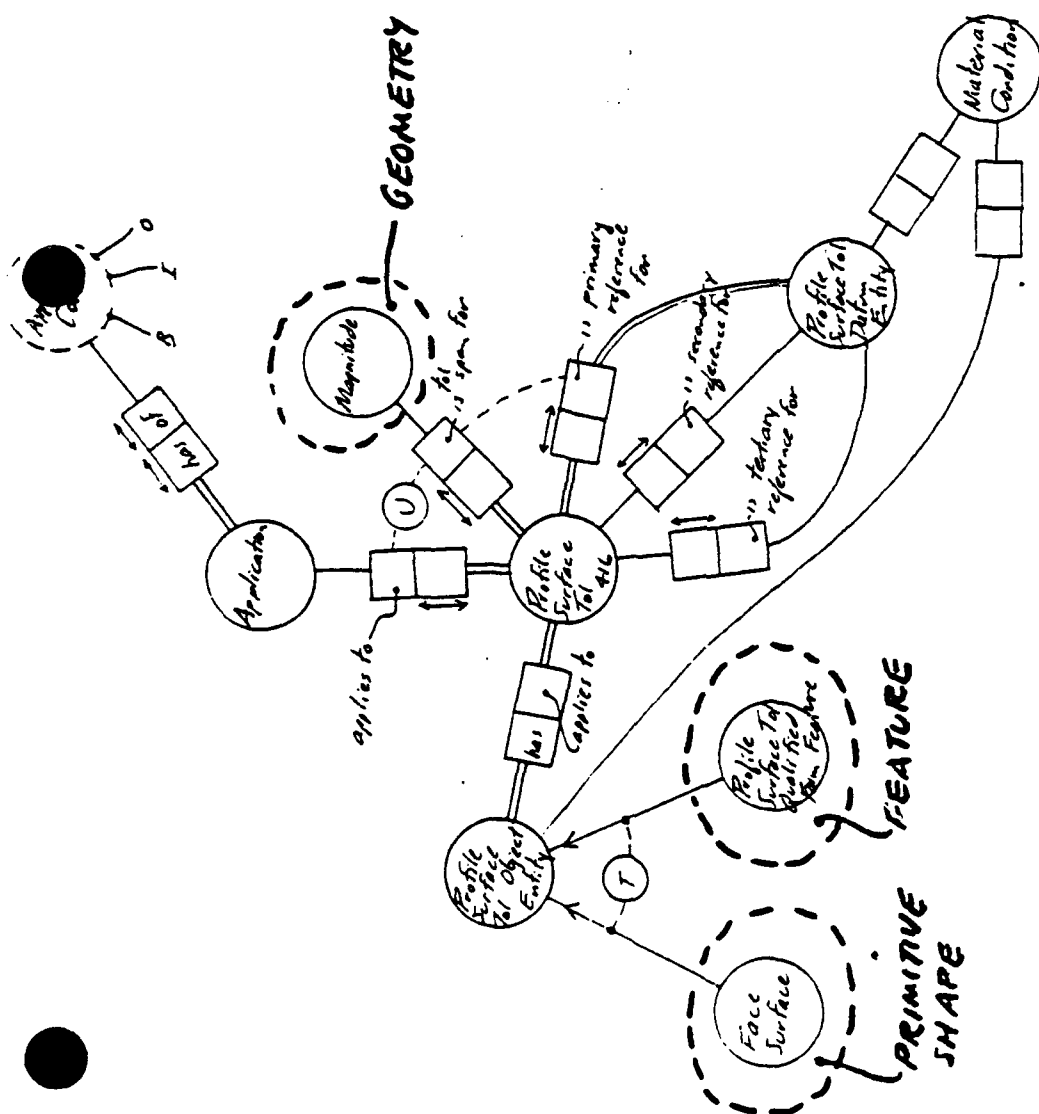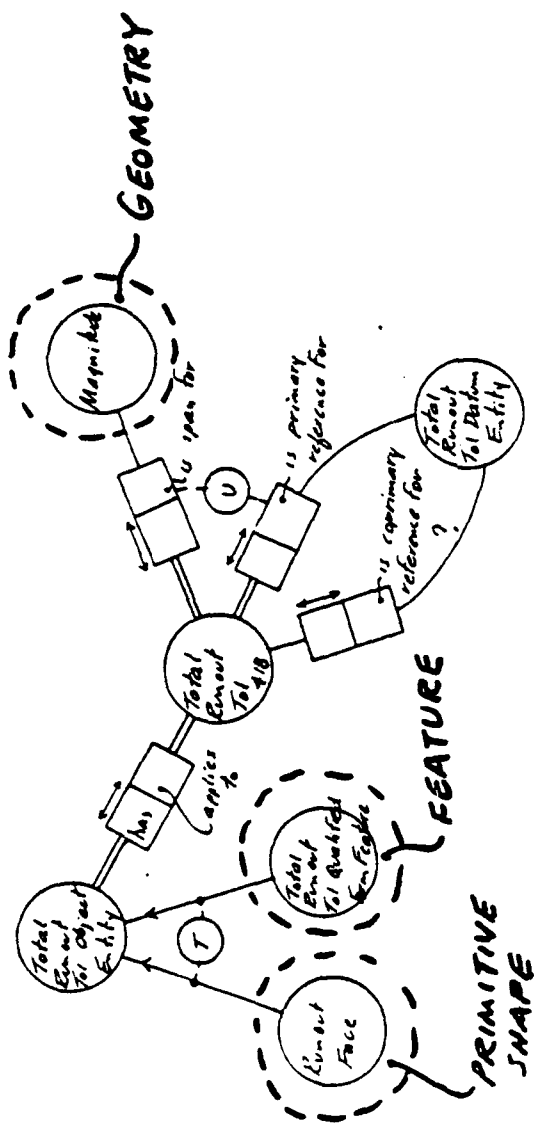
418 TOTAL RUNOUT TOLERANCE
GLOBAL MODEL

Appendix C5.4

Task 2

Tolerancing Specification Model

(DISCIPLINE)

```
-- START OF TOLERANCING ENTITIES
-- VERSION 0.1
-- AUTHOR COLSHURE/BURKETT

CLASS tolerancing OF
   (datum, tolerance);


CLASS datum OF
   (coordinate_toler_datum, geometric_toler_datum);


CLASS coordinate_toler_datum OF
   (angle_toler_datum, location_toler_datum);


CLASS geometric_toler_datum OF
   (angularity_toler_datum, circular_runout_toler_datum,
   concentricity_toler_datum, parallelism_toler_datum,
   perpendicularity_toler_datum, position_toler_datum,
   profile_line_toler_datum, profile_surface_toler_datum,
   total_runout_toler_datum);


CLASS tolerance OF
   (coordinate_toler, geometric_toler);


CLASS coordinate_toler OF
   (angle_toler, location_toler, size_toler);


CLASS geometric_toler OF
   (angularity_toler, circular_runout_toler, circularity_toler,
   concentricity_toler, cylindricity_toler, flatness_toler,
   parallelism_toler, perpendicularity_toler, position_toler,
   profile_line_toler, profile_surface_toler, straightness_toler,
   total_runout_toler);


CLASS angle_toler_datum OF
   (face, edge, feature-of-size);


ENTITY location_toler_datum;
NOROLE
   name : STRING;
ROLE
END;
```

```
ENTITY angularity_toler_datum;
NOROLE
  name : STRING;
ROLE
END;


ENTITY circular_runout_toler_datum;
NOROLE
  name : STRING;
ROLE
END;


ENTITY concentricity_toler_datum;
NOROLE
  name : STRING;
ROLE
END;


ENTITY parallelism_toler_datum;
NOROLE
  name : STRING;
ROLE
END;


ENTITY perpendicularity_toler_datum;
NOROLE
  name : STRING;
ROLE
END;


ENTITY position_toler_datum;
NOROLE
  name : STRING;
ROLE
END;


ENTITY profile_line_toler_datum;
NOROLE
  name : STRING;
ROLE
END;


ENTITY profile_surface_toler_datum;
NOROLE
  name : STRING;
ROLE
END;
```

```
ENTITY total_runout_toler_datum;
NOROLE
  name : STRING;
ROLE
END;


ENTITY angle_toler;
NOROLE
  name : STRING;
ROLE
  applies : LIST(1 to * ) OF REFER(angle_toler_object);
  toler_value : list(1 to 2) of t_magnitude;
  primary_datum : refer(angle_toler_datum);
  angle_direction : logical;
END;


ENTITY location_toler;
NOROLE
  name : STRING;
ROLE
  applies : LIST(1 to * ) OF REFER();
  toler_value : t_magnitude;
END;

entity location_toler;

ENTITY size_toler;
NOROLE
  name : STRING;
ROLE
  tolval : list(1 to 2) of t_magnitude;
  appies : LIST(1 to * ) OF REFER(size_toler_object);
END;


ENTITY angularity_toler;
NOROLE
  name : STRING;
ROLE
  applies : LIST(1 to * ) OF REFER(angularity_toler_object);
  toler_value : t_magnitude;
  primary_datum : refer(angularity_toler_datum);
  secondary_datum : OPTIONAL refer(angularity_toler_datum);
  tertiary_datum : refer(angularity_toler_datum);
  projected_toler_zone : refer(line);
END;
```

```
ENTITY circular_runout_toler;
NOROLE
  name : STRING;
ROLE
  applies : LIST(1 to * ) OF REFER(circular_runout_toler_object);
  toler_value : t_magnitude;
  primary_datum : refer(circular_runout_toler_datum);
  secondary_datum : OPTIONAL refer(circular_runout_toler_datum);
END;


ENTITY circularity_toler;
NOROLE
  name : STRING;
ROLE
  applies : LIST(1 to * ) OF REFER(circularity_toler_object);
  toler_value : t_magnitude;
END;


ENTITY concentricity_toler;
NOROLE
  name : STRING;
ROLE
  applies : LIST(1 to * ) OF REFER(concentricity_toler_object);
  toler_value : t_magnitude;
  primary_datum : refer(concentricity_toler_datum);
  secondary_datum : OPTIONAL refer(concentricity_toler_datum):
END;
  .


ENTITY cylindricity_toler;
NOROLE
  name : STRING;
ROLE
  applies : LIST(1 to * ) OF REFER(cylindricity_toler_object);
  toler_value : t_magnitude;
END;


ENTITY flatness_toler;
NOROLE
  name : STRING;
ROLE
  applies : LIST(1 to * ) OF REFER(flatness_toler_object);
  toler_value : t_magnitude;
  material_condition : t_material_condition;
  unit_square : ;
END;
```

```
ENTITY parallelism_toler;
NOROLE
  name : STRING;
ROLE
  applies : LIST(1 to * ) OF REFER(parallelism_toler_object;
  toler_value : t_magnitude;
  primary_datum : refer(parallelism_toler_datum);
END;


ENTITY perpendicularity_toler;
NOROLE
  name : STRING;
ROLE
  applies : LIST(1 to * ) OF REFER(perpendicularity_toler_object);
  toler_value : t_magnitude;
  primary_datum : refer(perpendicularity_toler_datum);
END;


ENTITY position_toler;
NOROLE
  name : STRING;
ROLE
  applies : LIST(1 to * ) OF REFER(position_toler_object);
  toler_value : t_magnitude;
  primary_datum : refer(position_toler_datum);
  secondary_datum : refer(position_toler_datum);
  tertiary_datum : refer(position_toler_datum);
END;


ENTITY profile_line_toler;
NOROLE
  name : STRING;
ROLE
  applies : LIST(1 to * ) OF REFER(profile_line_toler_object);
  toler_value : t_magnitude;
  primary_datum : refer(profile_line_toler_datum);
  secondary_datum : OPTIONAL refer(profile_line_toler_datum);
  tertiary_datum : refer(profile_line_toler_datum);
END;


ENTITY profile_surface_toler;
NOROLE
  name : STRING;
ROLE
  applies : LIST(1 to * ) OF REFER(profile_surface_toler_object):
  toler_value : t_magnitude;
  primary_datum : refer(profile_surface_toler_datum);
  secondary_datum : OPTIONAL refer(profile_surface_toler_datum):
  tertiary_datum : refer(profile_surface_toler_datum);
END;
```

```
ENTITY straightness_toler;
NOROLE
  name : STRING;
ROLE
  applies : LIST(1 to * ) OF REFER(straightness_toler_object);
  toler_value : t_magnitude;
END;


ENTITY total_runout_toler;
NOROLE
  name : STRING;
ROLE
  applies : LIST(1 to * ) OF REFER(total_runout_toler_object);
  toler_value : t_magnitude;
  primary_datum : refer(total_runout_toler_datum);
  secondary_datum : OPTIONAL refer(total_runout_toler_datum);
END;
```

Appendix C6

Task 2

Finite Element Modeling Discipline Model

C6.1   Discipline Model and Documentation

C6.2   Qualified Model (NIAM)

C6.3   Global Model (NIAM)

Appendix C6.1

Task 2

<u>Finite Element Modeling Discipline Model and Documentation</u>

(DISCIPLINE)

28

DEFINITIONS OF TERMS USED IN FEM IDEF MODEL

NOTE: These definitions were approved by the
IGES FEM Committee on Oct 16, 1985.

APPROVAL - This is an attribute of a model, refering to the
model's acceptance by the creating organization.

CONNECTIVITY CROSS REFERENCE (CR) - the assignment of a node to
an element. This refers to a single occurence of this cross
reference (CR), not a list of all the multiple occurences of
this entity.

COORDINATE SYSTEM - a frame of reference used to define the
location of a FEM or its components in 3D space.

COORDINATE SYSTEM TYPE - This is a label which is used to
distinguish between cartesian, cylindrical and spherical
coordinate systems.

CREATING SOFTWARE - The name of the software package used to
create the FEM model.

CREATION DATE - The date on which the FEM model was created.

CREATION TIME - The time of day that the model was created.
Time and date may be used to distinguished between similar models.

DEFINING COORDINATE SYSTEM - A coordinate system used to define
the location of a FEM entity. [See Comments at end of document]

DESCRIPTOR - Text associate with the model which can contain
any information about the model supplied by the creator.

ELEMENT - basic FEM building block defining the relationship
between its nodes.

ELEMENT/ENVIRONMENT CR - the assignment of an environment to an
element. This refers to a single occurence of this cross
reference, not a list of all the multiple occurence of this
entity.

ELEMENT/GROUP CR - assignment of an element to a group. This
refers to a single occurence w of this cross reference (CR), not
a list of all the multiple occurences of this entity.

ELEMENT/GEOMETRIC PROPERTY CR - assignment of a geometric
property to an element. This refers to a single occurence of
this cross reference (CR), not a list of all the multiple
occurences of this entity.

ELEMENT ORDER - the mathematical definition for the allowable deformation of an element edge. An element with nodes only at corners is linear or first order element. An element with an additional node on each edge between corner nodes is a parabolic or second order element. The order equals the number of non-corner nodes per element edge plus one. The element order and the element shape together imply an expected number of nodes to define the element. Missing nodes are caused by transition elements, and excess nodes imply face-located nodes.

ELEMENT PURPOSE DESCRIPTOR - Text which is used to describe the intended purpose for a particular element, and generally to define the limitations of the element capabilities. This may be important for element mapping between dissimiliar analysis codes. For example, a four-noded element could be a membrane element, a bending element, a shear panel, or a plate element with coupled bending and membrane behavior. This item will distinguish between geometrically similar element types.

ELEMENT SHAPE DESCRIPTOR - Text which is used to describe the shape of a particular element, i.e. wedge, brick, quad, triangle.

ENVIRONMENT -any external or internal influence on a FEM.

ENVIRONMENT/GROUP CR - assignment of an environment to a group. This refers to a single occurence of this cross reference (CR), not a list of all the multiple occurences of this entity.

ENVIRONMENT TYPE - A label that defines the type of environment being specified from a list of available types. For example, load or constraint.

FEM - Finite Element Model - a collection of elements and nodes that approximates a part or assembly; it may include geometric and material properties and an environment.

GEOMETRIC PROPERTY - values that may be used to describe the physical characteristics of an element. For example, shell element thickness, beam element area moment of inertia,etc.

GEOMETRIC PROPERTY TYPE - A label that defines the type of geometric property being specified from a list of available types. For example, shell, beam or composite layup properties.

GROUP - a collection of FEM nodes, FEM elements or FEM environments or any combination thereof.

LENGTH UNIT - the selected unit for specifying length throughout the model, such as inches or meters.

MASS UNIT - the selected unit for specifying mass throughout the model, such as kilograms or slugs.

MATERIAL PROPERTY - values that may be used to describe the constitutive nature of an element or a node.

MATERIAL PROPERTY TYPE - A label that defines the type of material property being specified from a list of available types. For example, mechanical, thermal or electrical properties.

NODE - a geometric location in a FEM; used to define elements and environments.

NODE/COORDINATE SYSTEM CR - The assignment of a coordinate system to a FEM node. This refers to a single occurence of this cross reference, not a list of all the multiple occurences of this entity.

NODE/ENVIRONMENT CR - the assignment of an environment to a node. This refers to a single occurence of this cross reference, not a list of all the multiple occurences of this entity.

NODE COORDINATE 1 - The first value from an ordered set of three values that define the spatial location of a node.

NODE COORDINATE 2 - The second value from an ordered set of three values that define the spatial location of a node.

NODE COORDINATE 3 - The third value from an ordered set of three values that define the spatial location of a node.

NODE/GROUP CR - assignment of a node to a group. This refers to a single occurence of this cross reference (CR), not a list of all the multiple occurences of this entity.

NODE/MATERIAL PROPERTY CR - assignment of a material property to a node. This refers to a single occurence of this cross reference, not a list of all the multiple occurences of this entity.

NODE SEQUENCE NUMBER - The number which represents the position of a node in a ordered list of nodes which defines an element, (the connectivity list). A single instance of the connectivity CR entity will use this sequence number to cross reference a node with a particular location on a particular element.

ORIGIN - The position in a coordinate system located with all zero coordinate values.

PART/FEM CR - the assignment of a FEM to a part. This refers to a single occurence of this cross reference (CR), not a list of all the multiple occurences of this entity.

TEMPERATURE UNIT - the selected unit for specifying temperature

throughout the model, such as degrees Centigrade or Rankine.

TRANSFORMATION MATRIX - The matrix used to specify the orientation and location of an entity in space, as well as the scale of the entity.

TIME UNIT - the selected unit for specifying time throughout the the model, such as seconds or hours.

RULES

1. **Any** change in a FE Model will result in a new model, and therefore, requires a new FEM ID.

2. Any sequence number in the FEM connectivity entity greater than those indicated by order and shape must refer to an excess node or several excess nodes.

COMMENTS AND SUGGESTIONS

1. The "defining coordinate system" is only used in the NQDE entity, and probably should be replaced there by "coordinate system" since the defining coordinate system definition and the coordinate system definition are essentially identical. This means changing the non-key attribute name in the NODE entity and eliminating the definition from the data dictionary.

2. The PART/FEM CR entity may or may not be appropriate, since we removed the PART definition from this data dictionary.

W. R. Freeman
Allied Signal Bendix Aerospace
Oct 1985/corrections Jan 1986

Finite Element Modeling Relationships

Natural Language Version of IA Model, Derived from
IDEF Model created by IGES FEM Subcommittee

NOTE: These statements are only valid when the terms
used are defined as per the FEM IDEF Model
dictionary of terms.

1. A defining coordinate system may have many nodes.

2. An approval may be for many models.

3. An author may create many models.

4. A connectivity must always have only one node sequence
   number.

5. A connectivity must always have only one node.

6. A connectivity must always refer to only one element.

7. A coordinate system may be referenced by many node/coordinate
   system CRs.

8. A coordinate system type may refer to many coordinate systems.

9. A coordinate system must always have only one coordinate system.
   type

10. A coordinate system must always have only one transformation
    matrix.

11. A coordinate system must always have only one origin.

12. A coordinate system may be referenced by many material
    properties.

13. A coordinate system must always refer to only one model.

14. An origin may refer to many coordinate systems.

15. Creating software may refer to many models.

16. A creation date may refer to many models.

17. A creation time may refer to many models.

18. A descriptor may refer to many models.

19. An element/environment CR must always refer to only one
    environment.

20. An element/environment CR must always refer to only one element.

21. An element/geometric property CR must always refer to only one geometric property.

22. An element/geometric property CR must always refer to only one element.

23. An element/group CR must always refer to only one group.

24. An element/group CR must always refer to only one element.

25. An element/material property must always refer to only one material property.

26. An element/material property must always refer to only one element.

27. An element order may refer to more than one element.

28. An element must always have only one purpose descriptor.

29. An element must always have only one element order.

30. An element must always have only one shape descriptor.

31. An element may be refered to by many element/environment CRs.

32. An element may be refered to by many element/material property CRs.

33. An element may be refered to by many element/geometric property CRs.

34. An element may be refered to by many element/group CRs.

35. An element must always be referred to by many connectivities.

36. An element must always refer to only one FEM model.

37. An element must always have only one element number.

38. An environment/group CR must always refer to only one environmen

39. An environment/group CR must always refer to only one group.

40. An environment type may be used by many environments.

41. An environment may refer to many element/environment CRs.

42. An environment may be referenced by many environment/group CRs.

43. An environment must always have only one environment type.

44. An environment may be referenced by many node/environment CRs.

45. An environment may be referenced by many element/environment CRs.

46. An environment may include restraints.

47. An environment may include constraints.

48. An environment may include temperatures.

49. An environment may include loads.

50. A geometric property type may refer to many geometric properties.

51. A geometric property must always have only one geometric property type.

52. A geometric property may include beam geometric properties.

53. A geometric property may be referenced by many element/geometri property CRs.

54. A geometric property may include a composite layup.

55. A geometric property may include point geometric properties.

56. A geometric property may include shell geometric properties.

57. A group type may refer to many groups.

58. A group must always be of only one group type.

59. A group may have only one node/group CR.

60. A group may have only one element/group CR.

61. A group may have only one environment/group CR.

62. A group may include a region.

63. A group may include a color.

64. A group must always refer to only one model.

65. A length unit may be used in many models.

66. A mass unit may be used in many models.

67. A material property type may refer many material properties.

68. A Material property must always have only one material property type.

69. A material property must always reference zero or one coordinate system.

70. A material property may be referenced by many node/material property CRs.

71. A material property may refer to many element/material property CRs.

72. A material property may include electrical properties.

73. A material property may include acoustical properties.

74. A material property may include thermal properties.

75. A material property may include mechanical properties.

76. Node/environment CRs must always refer to only one environment.

77. A node/environment CR must always refer to only one node.

78. Node/material property CR must always refer to only one material property.

79. A node/material property CR must always refer to only one node.

80. Node/coordinate system CRs must always refer to only one node.

81. A node/coordinate system CR must always reference only one coordinate system.

82. Node coordinate 1 may refer to many nodes.

83. Node coordinate 2 may refer to many nodes.

84. Node coordinate 3 may be used by many nodes.

85. A node/group CR must always refer to only one group.

86. A node/group CR must always refer to only one node.

87. A node sequence number may be referenced by many connectivities.

88. A node may reference many node/coordinate system CRs.

Appendix C6.2

Task 2

Finite Element Modeling Qualified Model (NIAM)
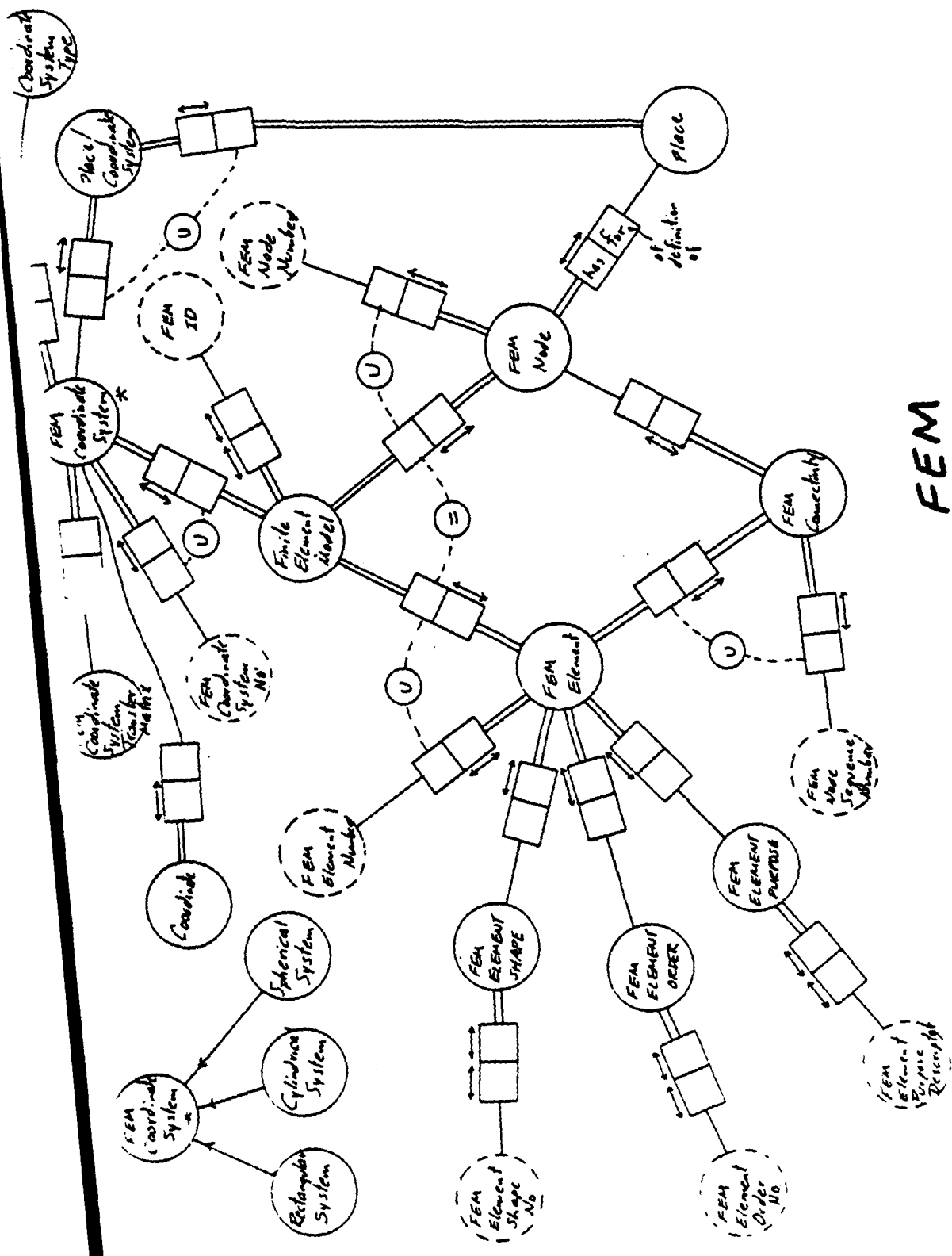
(DISCIPLINE)

FEM QUALIFIED

FEM QUALIFIED

FEM
QUALIFIED

FEM
QUALIFIED

FEM
QUALIFIED
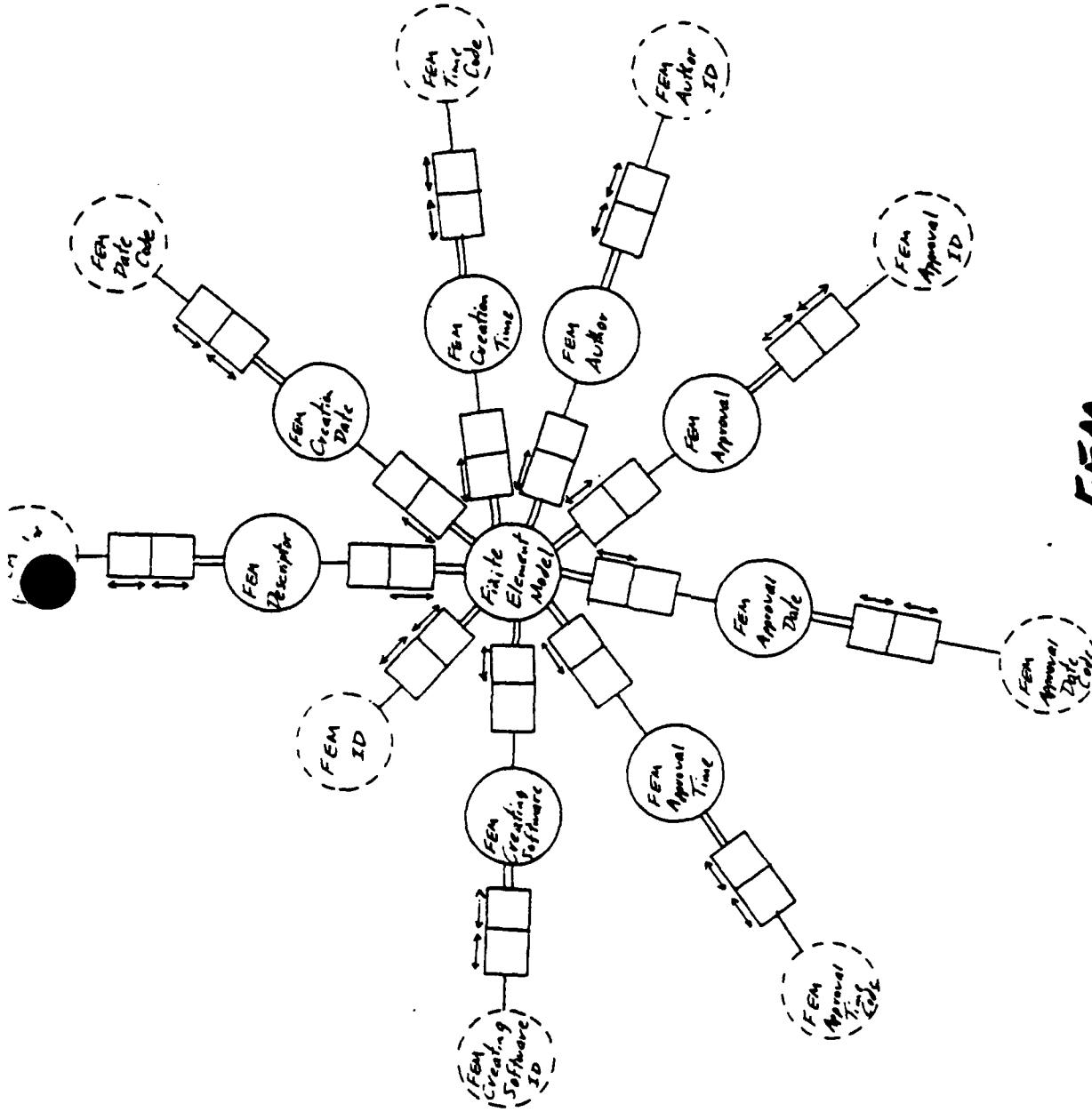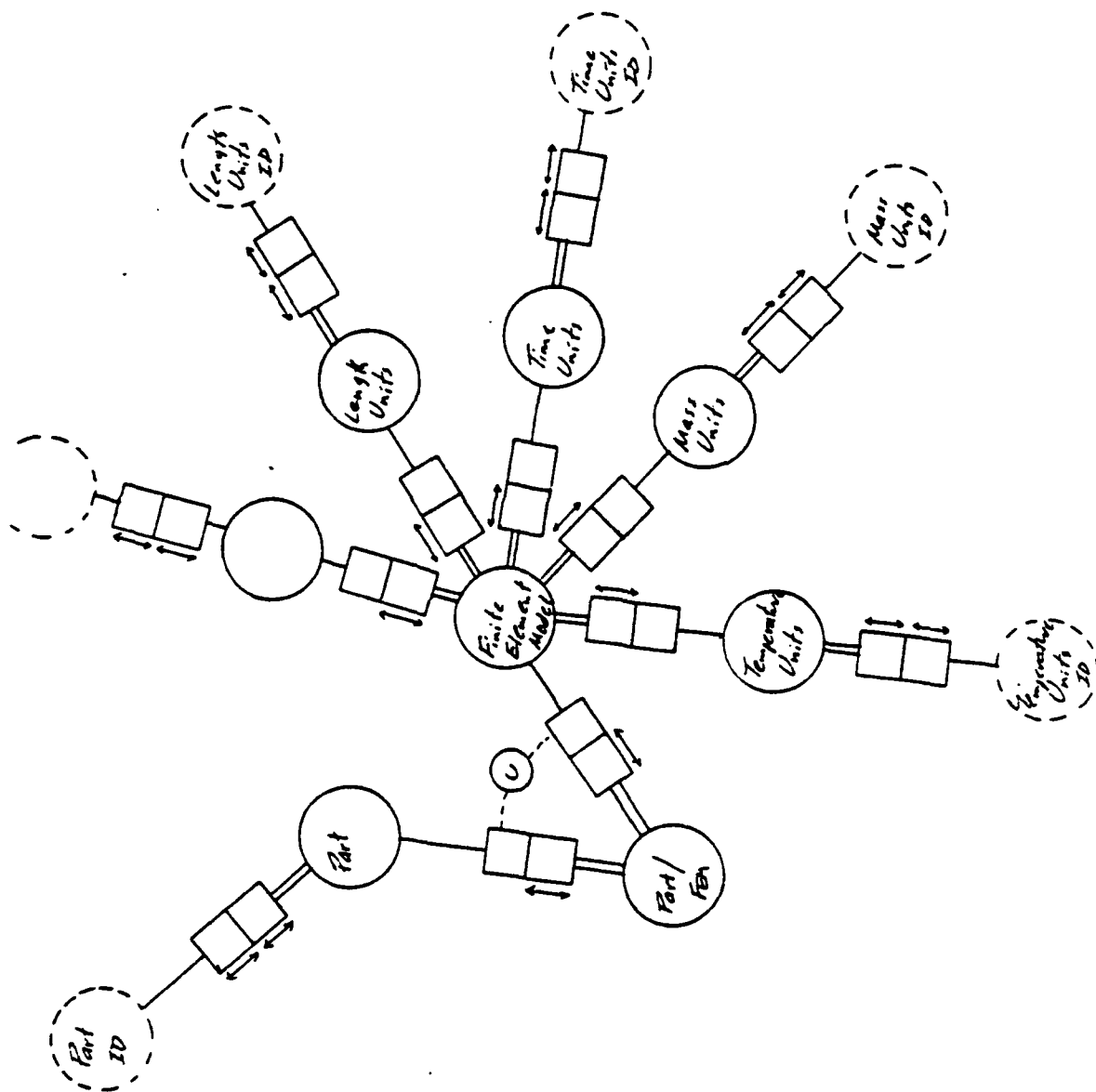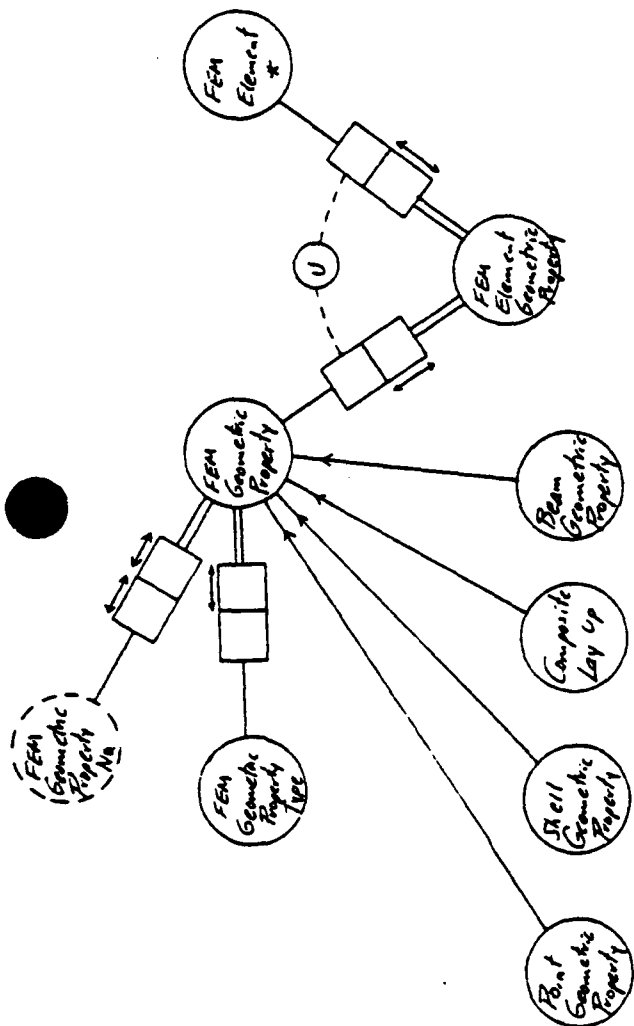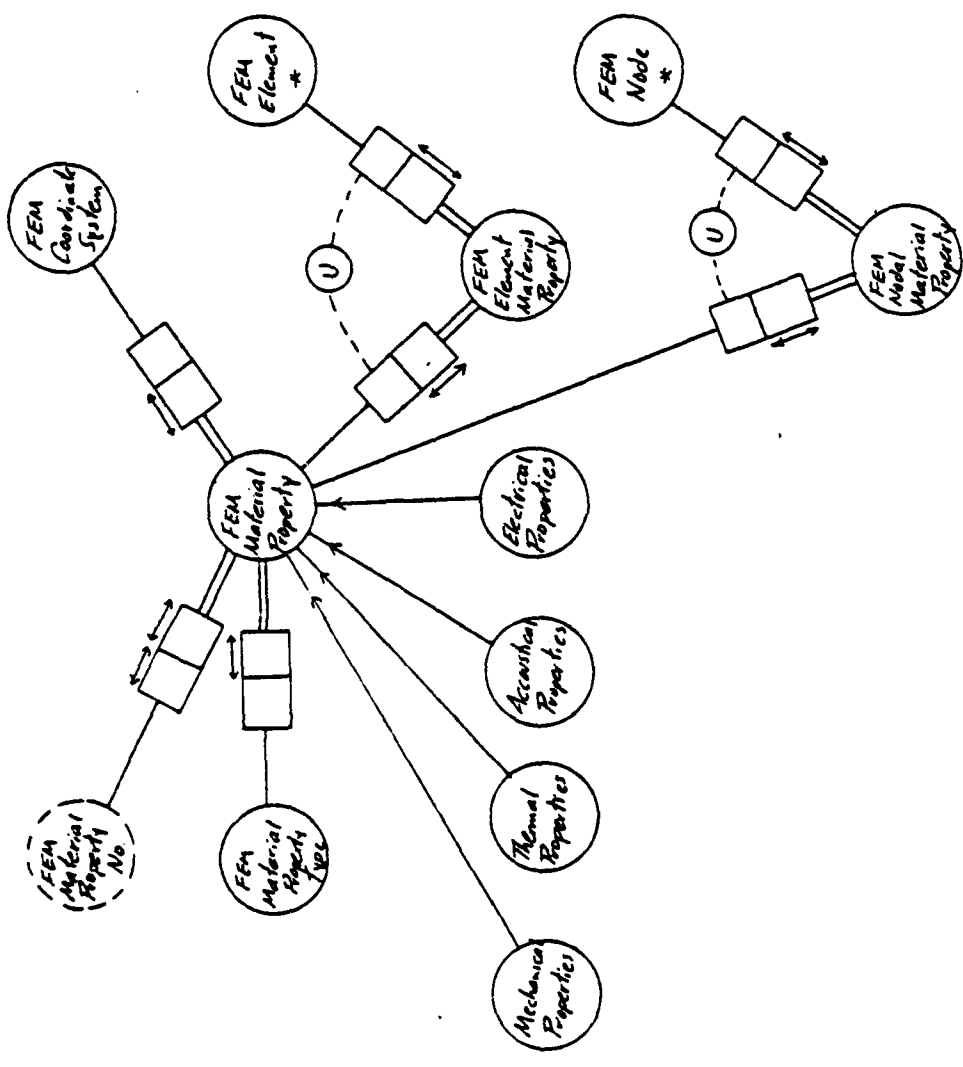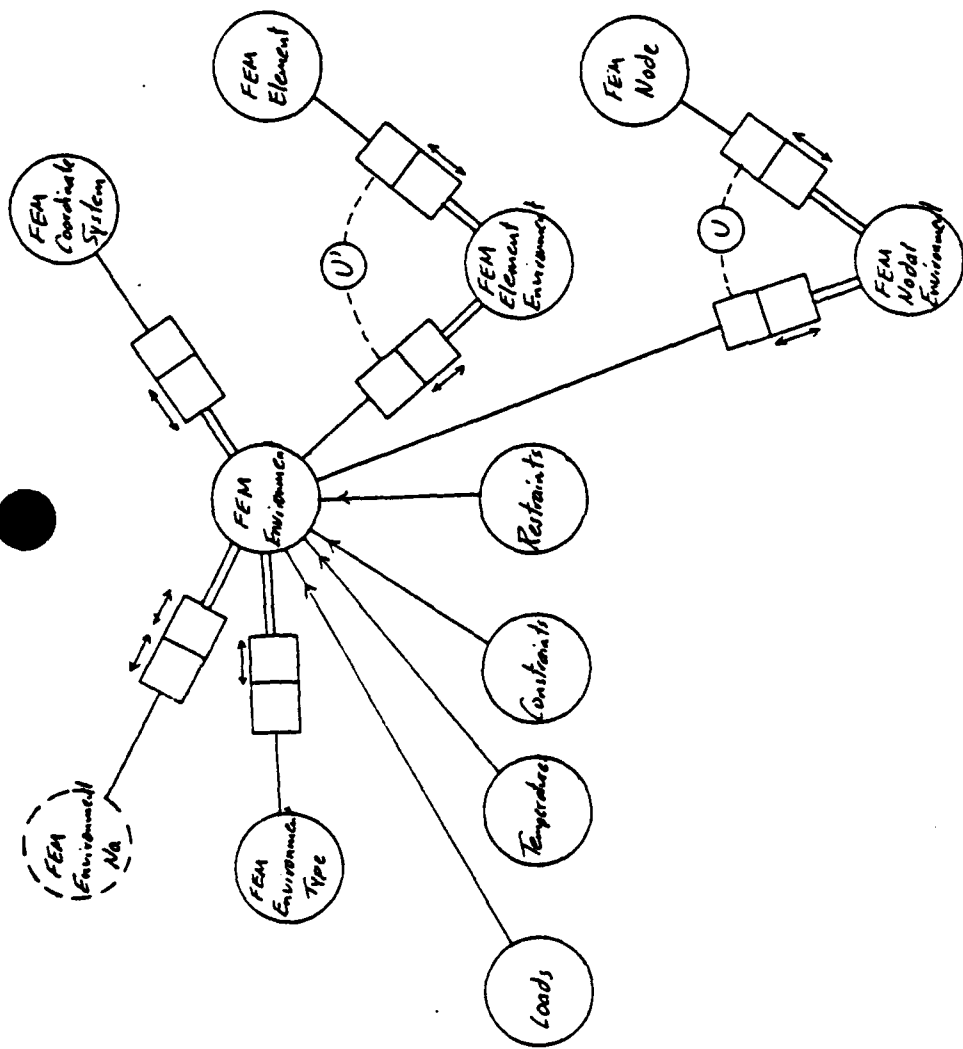
FEM
QUALIFIED
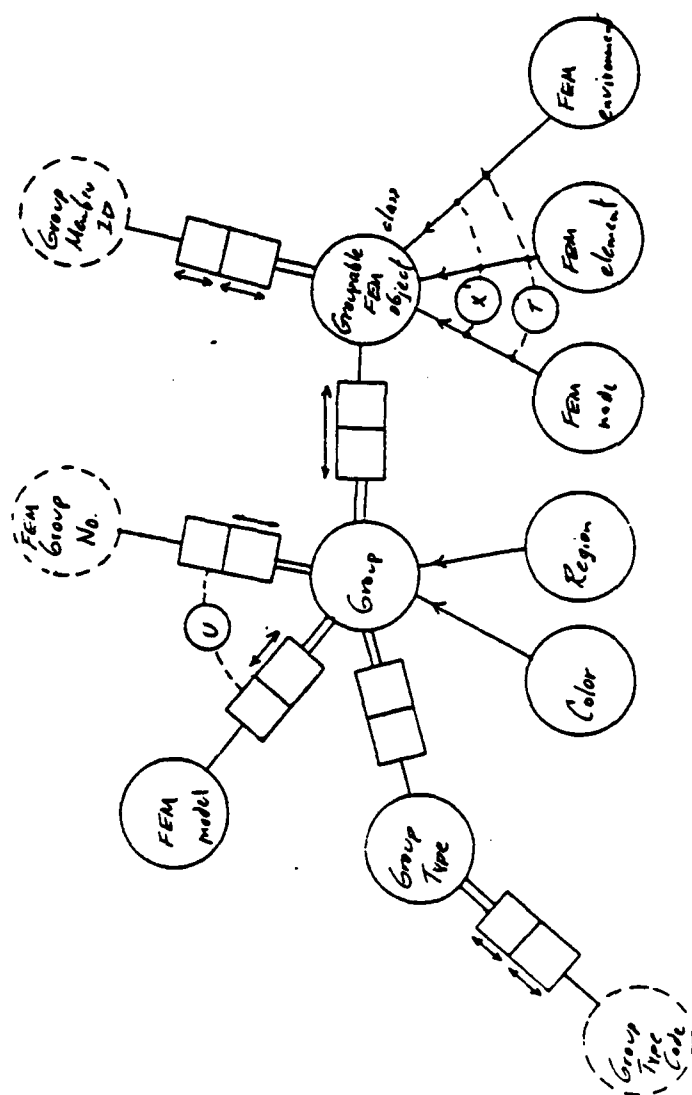
FEM QUALIFIED

Appendix C6.3

Task 2

Finite Element Modeling Global Model (NIAM)

(DISCIPLINE)

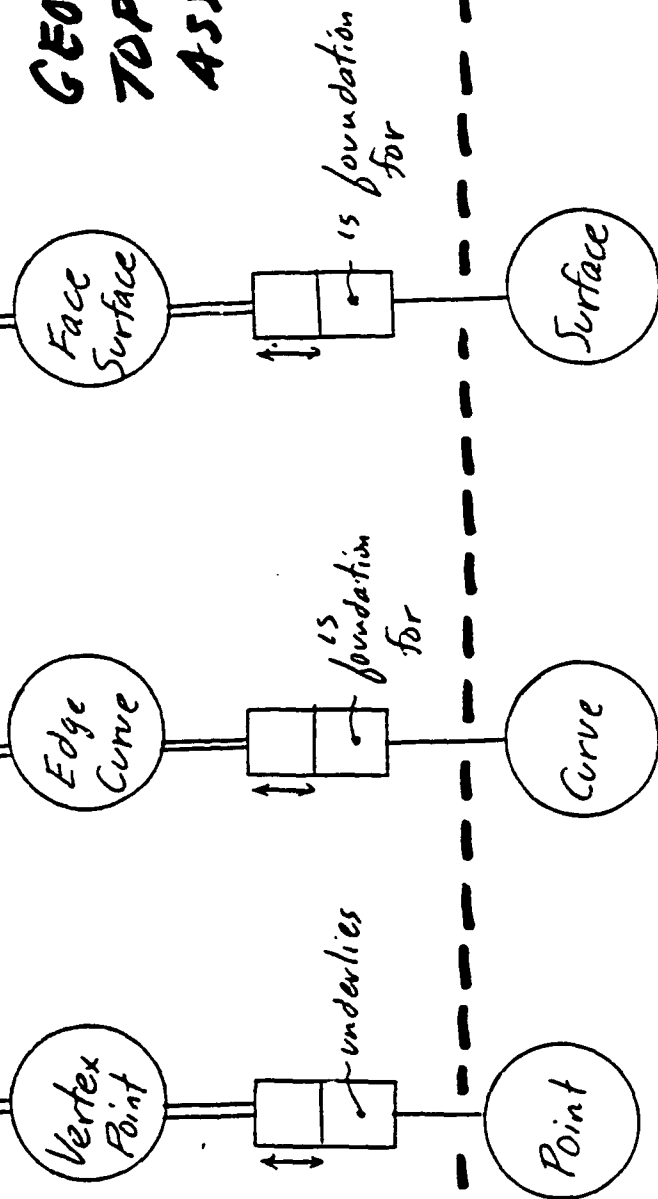FEM GLOBAL

FEM Node Number

FEM IP

Place *

FEM Element Number

FEM Element Shape No.

FEM Element Order No.

FEM Element Collapse Principle

FEM Node

Finite Element Model

FEM Element

Vertex

Edge

Place *

FEM Element Shape

FEM Element Order

FEM Element Purpose

Transversal

Direction

Vertex Number

has location of

vertices location of

is a collection of

shares

defines

defined as grid?

comprises

defines location of

has location at

is composed of

has connectivity defined by

defines connectivity for

has connectivity

is imposed on

FEM Element Note 1:

of has

of has

of has

of has

of has

of has

U

U

U

U

U

Note: all element shape is buried in this object. Diagram shows Global topology. Global topology is determined from explicit element geometry and association of FEM vertices to FEM node.

FEM Node (collection of vertex, as vertices of el)

Vertex

FEM Node (collection of vertex of)

FEM Element

FEM Element

edge

Figure 4-19

TOPOLOGY

GEOMETRY

Vertex — defined by — Vertex Point — underlies — Point

Edge — bounds — Edge Curve — is foundation for — Curve

Face — bounds — Face Surface — is foundation for — Surface

GEOMETRY TOPOLOGY ASSOCIATIVITY (GTA)*

* Also referred to as "Primitive Skive"

Appendix C8

Task 2

Geometry-Topology Associativity Resource Model (NIAM)

(RESOURCE)

```
-- TOPOLOGY SCHEMA
-- VERSION  0.0

CLASS topology OF
    (vertex, edge, loop, face, shell, object);


ENTITY vertex;
NOROLE
   name : OPTIONAL string;
ROLE
   basepoint : REFER(point);
END;


ENTITY edge;
NOROLE
   name : OPTIONAL string;
ROLE
   basecurve : REFER(curve);
   limit0 : REFER(vertex);
   limit1 : REFER(vertex);
END;


ENTITY loop;
NOROLE
   name : OPTIONAL string;
ROLE
   member : LIST(1 TO *) OF REFER(edge);
END;


ENTITY face;
NOROLE
   name : OPTIONAL string;
ROLE
   basesurface : REFER(surface);
   boundloop : REFER(loop);
   interiorloop : LIST(0 TO *) OF REFER(loop);
END;


ENTITY shell;
NOROLE
   name : OPTIONAL string;
ROLE
   member : LIST(1 TO *) OF REFER(face);
END;
```

```
ENTITY object;
NOROLE
  name : OPTIONAL string;
ROLE
  boundshell : REFER(shell);
  interiorshell : LIST(O TO *) OF REFER(shell);
END;
```

Topology

Appendix C7.2

Task 2

<u>Topology Resource Model (DSL)</u>

(RESOURCE)

Appendix C7

Task 2

<u>Topology Resource Model</u>

(RESOURCE)

C7.1 NIAM Model

C7.2 DSL Model

Appendix C7.1

Task 2

Topology Resource Model (NIAM)

(RESOURCE)
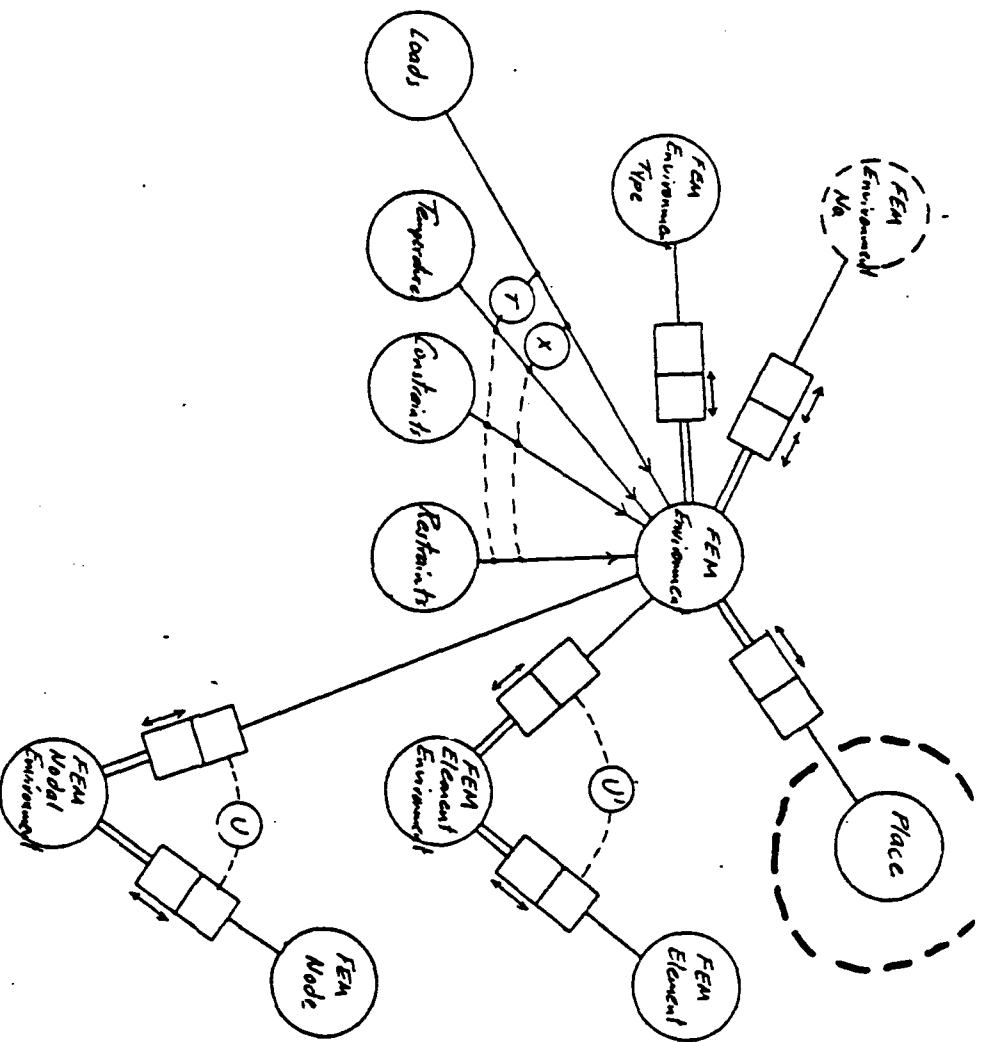
FEM GLOBAL

FEM
GLOBAL

Appendix D

Primers

D1.   IDEF-1 And IDEF-1 Extended (IDEF1-X)

D2.   Nijssen Information Analysis Method (NIAM)

D3.   Data Specification Language (DSL)

Appendix D1

IDEF1 And IDEF1-X

36

A BRIEF OVERVIEW OF THE
IDEF1X MODELING METHODOLOGY

Prepared by Roger Gale

In this Appendix, references to Conceptual Schema, External Schema and Internal Schema are based upon the concepts advanced by the ANSI/X3/SPARC committee.

IDEF1X is the latest version of the Data Modeling methodology developed under the USAF ICAM project. Because of the USAF ICAM contracts, it is probable that more product data modeling has been performed in IDEF1X and its original version, IDEF1, than in any other methodology.

It is important that the reader understand that IDEF1X is a language with syntax and semantics for expressing a data model plus a discipline, or method, for developing the model. A data model representing a portion of the Conceptual Schema of an enterprise may only be reached if the discipline is applied. The modeling language in and of itself does not guarantee a Conceptual Schema model. Persons using the language without understanding and applying the discipline tend to model portions of an External Schema or Internal Schema.

The developers of IDEF1X assigned overriding importance to its use to communicate among humans. There is a belief that a Conceptual Schema demands considerable review by people before the rules embodied in it have been validated. Therefore, the language consists of graphics and text with the most semantically important ideas being given graphic importance as an aid to that necessary human communication.

The largest text portion of the model is the Glossary in which all names of entities and attributes are defined. Another text portion contains identification of constraints which are not shown graphically such as Data Types and Domains for Attributes, and Path Assertions.

Part 1 of this appendix is primarily a brief explanation of the discipline for developing an IDEF1X conceptual model. Part 2 is an explanation of the graphic syntax and semantics. Part 3 explains the differences between IDEF1 and IDEF1X.

The discipline for IDEF1X is a series of steps beginning with definition of the areas of the enterprise which are of interest and continuing through a series of specific refinements to a fourth normal form, relational model. Within the scope of interest, this is a "top down" methodology. It permits modelers to begin with a model of considerable abstraction and progress in steps to the desired degree of detail.

In the paragraphs which follow, the sequence of IDEF1X modeling steps is outlined very briefly. The reader should not expect to be able to model adequately from this material. A detailed text can be obtained from the USAF. For information about obtaining a copy it is suggested that the following person be contacted:

David L. Judson
AFWAL/MLTC
Bldg. 653, Area B, Room 221
Wright Patterson AFB, Ohio 45433-6533

(513) 255-6976

The modeling procedure has five phases. The initial phase is Phase 0. In phase 0, the scope and objectives of the modeling project are established. One of the key factors is to define whether the model is to represent an "AS-IS" condition or a "TO-BE" condition.

Phase 1 involves the initial defintion of Entities. Phase 2 defines the relationships among the entities. Phase 3 defines keys for each entity. In Phase 4, non-key attributes are defined and final refinement of the model is performed.

PHASE 1

The first step is to create an Entity Pool. An Entity is defined as a kind of thing about which the enterprise keeps data. The thing may be concrete such as an employee or a machine, or an abstract thing such as a schedule event. There are tests for entities. Each instance of an entity must have a unique identifier. For example, each employee has a unique employee number assigned by the enterprise. There should be attributes (properties or characteristics) of the entity which represent data the enterprise maintains. In the employee case, the enterprise normally keeps data about such attributes as name, birth date, employment date, etc.

Each candidate entity must be given a definition and a name. There may be a variety of names in common use in the enterprise which are applied to the entity. However, in order that the model will have stability, a specific definition and name must be chosen as the Conceptual name. A basic rule for a conceptual model is "same name/same meaning".

PHASE 2

The next step is to discover the direct relationships among the entities. It is recommended that this be started by preparing a matrix with the list of candidate entities on each axis. Then each pair of entities is examined to determine if there is a direct relationship between them. If there is, an "X" is placed at that intersection of the matrix. It is useful here if sentences are written stating how the entities are related. For example:

> During his tenure with the company an Employee is assigned to one or more Departments.

> A Department has many assigned Employees.

From the entity relationship matrix, an initial entity-relationship diagram is drawn. This diagram uses the entity boxes and relationship lines, labels and cardinality symbols shown in Part 2 of this appendix. The relationships are defined and labelled. At this stage, many of the relationships will be non-specific (i.e. many-to-many). The definition of the realtionship between two entities must be examined in both directions. Determine how many instances of one entity can be related to one instance of the other entity in each direction. The relationship must be labeled. If it is non-specific, the relationship must be labeled to describe the relationship in each direction. A specific relationship only requires a label reading from parent to child.

Here it is important to remember that it is the data of the enterprise which is being modeled not a computer system. Computer systems have frequently been developed so that their data files contain only the current knowledge. Meanwhile, the enterprise keeps copies of former states of knowledge. A computer system may only show the name of the current General Manager of a Division. The records of the Division show all previous General Managers. Consequently, the enterprise data model should show that a Division has one or many General Managers.

PHASE 3

The objectives of Phase 3 are to:

Refine the non-specific relationships from Phase 2.

Define key attributes for each entity.

Migrate primary keys from parent entities to child entities to establish foreign keys.

Validate relationships and keys.

The next step is to resolve all non-specific relationships. The non-specific relationships from the initial diagram must be refined into existence-dependency (parent-child) relationships or generalization relationships (see Part 2 for examples). The process of refining non-specific relationships converts each non-specific relationship into two specific relationships. New entities evolve out of this process. Each of the new entities must be provided a name and defintion in the Glossary. Entities resulting from this process are informally referred to as "derived entities". A derived entity tends to be more abstract than the initial business entities.

When specific realtionships have been defined, it is time to establish "keys" for each entity. A key is an attribute, or concatenation of two or more attributes, which will uniquely identify a single occurrence of an entity. Candidate keys are developed and defined for each entity.

When an entity has more than one candidate key, one must be established as the primary key and the others identified as alternate keys. Alternate keys are shown below the line in the entity box and identified by placing (AKn) following their name (see Part 2 examples). Because there may be more than one alternate key, the "n" is a number indicating which attributes belong to which alternate key. Keys are an attribute, or combination of attributes, of the entity. Each of the key attributes must be provided a definition and name in the model glossary.

The Primary Key of each parent entity must be migrated to each of its child entities. It may migrate to become part of the key of a child, or it may migrate as a non-key attribute of the child. In either case, it is shown in the child and identified as a "foreign key" by placing (FK) after its name in the child. Role-names are assigned to migrated keys as required (see Part 2 role name examples).

Key attributes must be validated by testing that they conform to several rules as follows:

1. No attribute of an entity may have more than one value for each instance of the entity (No-Repeat rule)

2. No attribute of an entity may have a null value for any instance of the entity (No-Null rule).

3. Where an entity has a compound key, it must not be possible to split that entity into multiple entities with simpler keys (Smallest-Key rule).

4. Where there are dual relationship paths between two entities, assertions must be made as to whether both paths from the dependent entity reach the same or different instances of the independent entity (Path Assertions).

PHASE 4

After the relationships have been refined and keys established, the next step is to determine the non-key attributes of each entity. For each entity, candidate attributes are listed. These represent the properties or characteristics of each entity about which the enterprise maintains data. Each candidate attribute must be given a definition and name in the model glossary.

Each candidate attribute is assigned to an entity, its "owner". Attributes are then tested with the "No-Repeat" and "No-Null" rules. Attributes which viloate the rules require the introduction of new "derived" entities for their ownership. These new entities must be named and defined in the Glossary, their appropriate relationships defined and keys migrated to them.

An additional rule must be applied now. It is applied to attributes of entities with compound keys. This is the Full-Functional-Dependency rule. This rule requires that no owned attribute value of an entity instance can be identified by less than the entire key value for the instance. The correct application of this rule is equivalent to the second normal form of relational theory.

A final rule is the No-Transitive-Dependency rule. This rule requires that no owned, non-key attribute value of an entity instance can be identified by the value of another owned, or inherited, non-key value of an attribute of the entity instance. This rule is equivalent to third normal form in ralational theory.

These last two rules can be summed up in the statement; "a non-key attribute must be dependent upon the key, the whole key and nothing but the key".

SUMMARY

The IDEF1X modeling methodology is a "top down" method which starts with abstraction and progresses to required detail. It utilizes the Entity idea to collect data (Attributes) in the manner in which they appear in the enterprise being modeled. When the model is fully refined according to the rules, each entity represents a statement that values for all of the attributes of that entity appear at the same time and place within the enterprise.

The IDEF1X modeling language and discipline, when used by trained modelers, and properly validated will result in the definition of a stable Conceptual Schema For PDES, instability in the Logical Layer Conceptual Schema will result in compatbility problems among versions which is an undesirable characteristic.

The IDEF1X language is not only useful for modeling a Conceptual Schema. It may also be used as a planning tool by preparing models of high degrees of abstraction to illustrate fundamental data relationships in an enterprise. The next two pages are an example of such a "Data Planning Model". This model is at a very high degree of abstraction. For example, I have been working with an enterprise where there is a model containing about 300 entities which are represented by the handful of entities on the second page. Such models are used to establish scopes of projects by examining data affinities. This model probably represents a significant percentage of the potential future coverage of PDES.

PRODUCT DATA PLANNING MODEL

REVISED 21 APRIL 1988
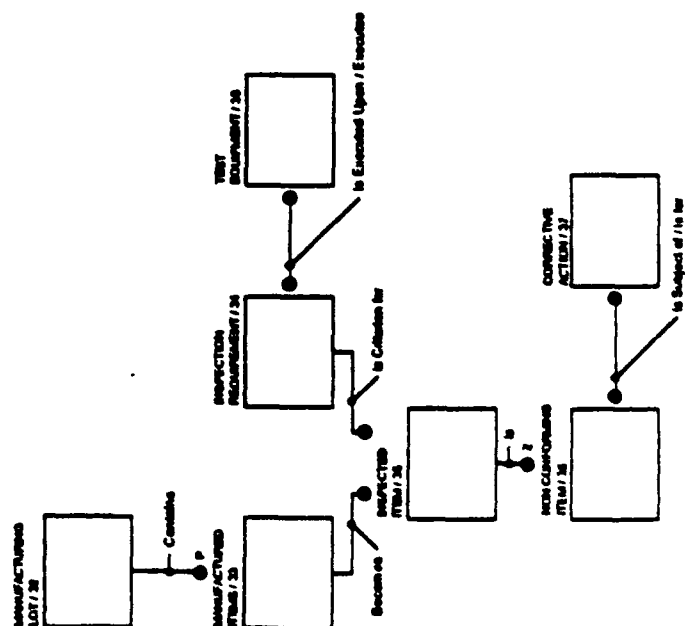
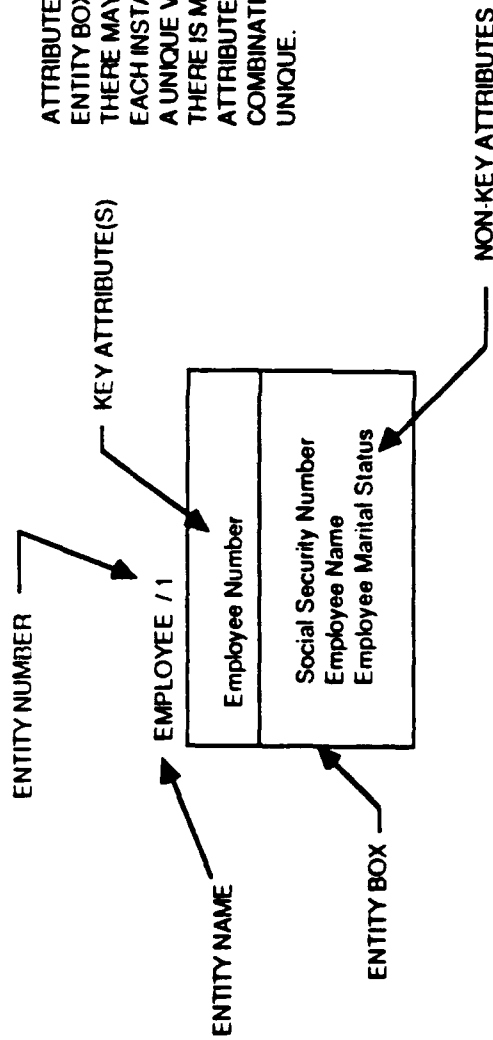# NOTICE

IDEF1X

READER'S REFERENCE

THE ENTITY IDEA

- AN ENTITY IS A THING ABOUT WHICH THE ENTERPRISE KEEPS DATA

- AN ENTITY MAY BE A REAL, PHYSICAL THING LIKE AN EMPLOYEE OR A MACHINE; OR IT MAY BE AN ABSTRACT THING SUCH AS A SCHEDULE EVENT

- AN ATTRIBUTE IS DATA MAINTAINED BY THE ENTERPRISE WHICH REPRESENTS A PROPERTY, OR CHARACTERISTIC, OF THE ENTITY IN WHICH THE ENTERPRISE IS INTERESTED
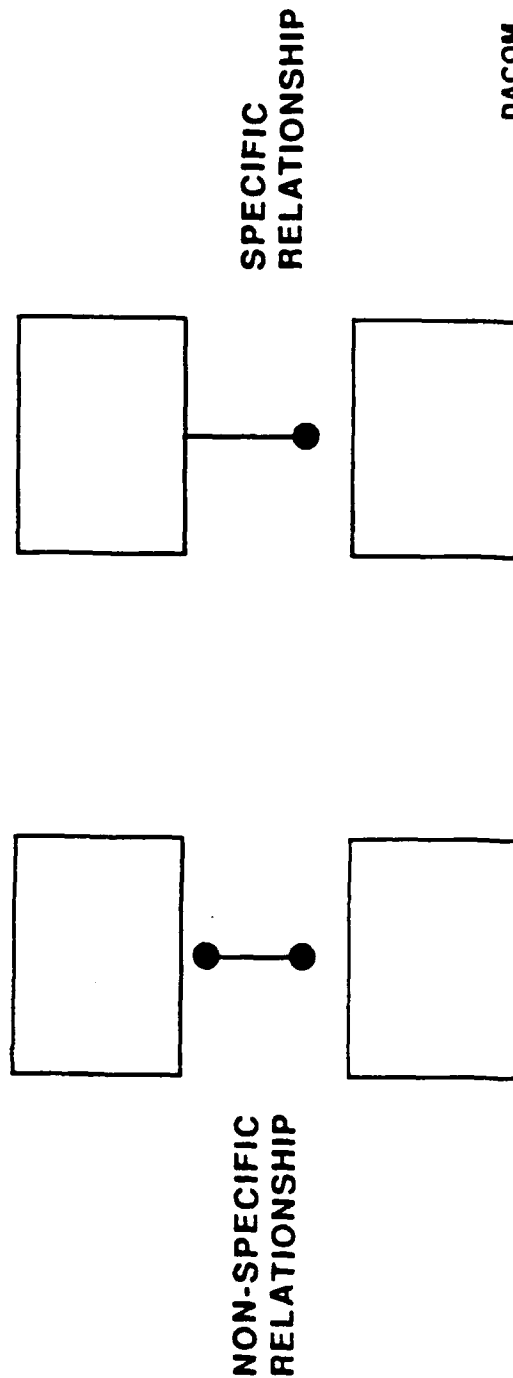
324 2
[1]1

THE ENTITY CONSTRUCT

ENTITY NUMBER

KEY ATTRIBUTE(S)

ENTITY NAME

ENTITY BOX

NON-KEY ATTRIBUTES

EMPLOYEE / 1

Employee Number

Social Security Number
Employee Name
Employee Marital Status

ATTRIBUTES ABOVE THE LINE IN THE ENTITY BOX FORM THE ENTITY KEY. THERE MAY BE MORE THAN ONE. EACH INSTANCE OF THE ENTITY HAS A UNIQUE VALUE OF THE KEY. WHEN THERE IS MORE THAN ONE ATTRIBUTE AS KEY, IT IS THE COMBINATION OF VALUES THAT IS UNIQUE.

IN A FULLY REFINED MODEL, EACH NON-KEY ATTRIBUTE HAS EXACTLY ONE VALUE FOR EACH INSTANCE OF THE ENTITY. TWO DIFFERENT INSTANCES OF THE ENTITY MAY HAVE IDENTICAL VALUES FOR NON-KEY ATTRIBUTES.

DACOM
(c) D. Appleton Co. Inc

324 3
10)1

RELATIONSHIP CONCEPTS

- A RELATIONSHIP ESTABLISHES THE FACT THAT AN ENTITY IS PAIRED WITH ANOTHER ENTITY IN SOME MANNER

- RELATIONSHIPS MAY BE "SPECIFIC" OR "NON-SPECIFIC"

- IN A FULLY REFINED MODEL, ONLY "SPECIFIC" RELATIONSHIPS ARE PERMITTED

- RELATIONSHIPS ARE INDICATED BY LINES AND DOTS BETWEEN ENTITIES

SPECIFIC
RELATIONSHIP

NON-SPECIFIC
RELATIONSHIP

EXISTENCE DEPENDENCY

PARENT ENTITY

EMPLOYEE / 1

Employee Number

has

CHILD ENTITY

DEPENDENT / 2

| Employee Number(FK)
Dependent Name |
|---|
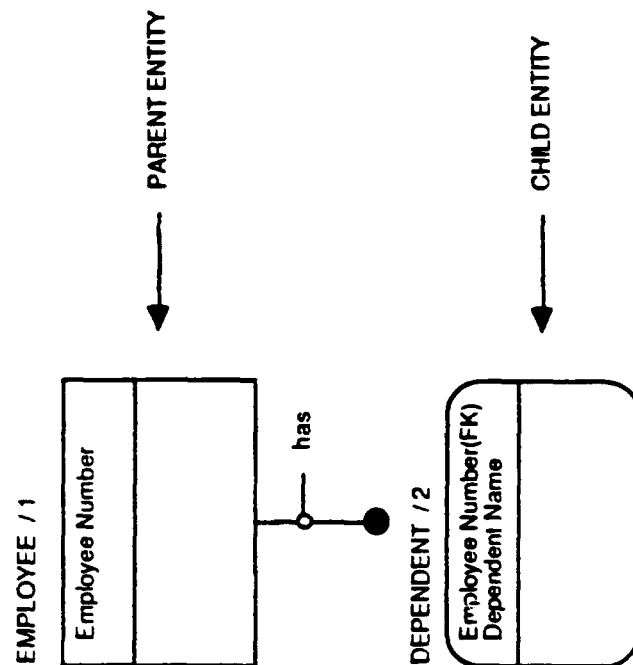
IN EVERY SPECIFIC RELATIONSHIP
THERE IS A PARENT ENTITY AND A
CHILD ENTITY. THE CHILD ENTITY IS
DEPENDENT FOR ITS EXISTENCE UPON
ITS PARENT. WHEN THE CHILD HAS
MORE THAN ONE PARENT IT IS
EXISTENCE-DEPENDENT UPON ALL OF
ITS PARENTS.

THIS MEANS THAT NO INSTANCE OF
THE CHILD ENTITY MAY OCCUR
WITHOUT A RELATED INSTANCE OF
EACH OF ITS PARENTS.

3245,
1101

CARDINALITY

SYMBOL	MEANING
●	ZERO, ONE OR MANY
● z	ZERO OR ONE
● p	ONE OR MANY
● n	A SPECIFIC QUANTITY (n)

RELATIONSHIP CARDINALITY EXAMPLES

NON-SPECIFIC -

SPECIFIC -

3247
DD1
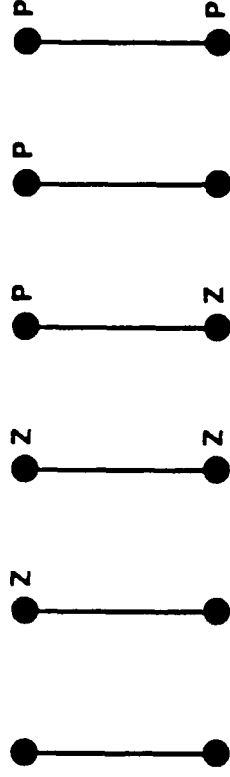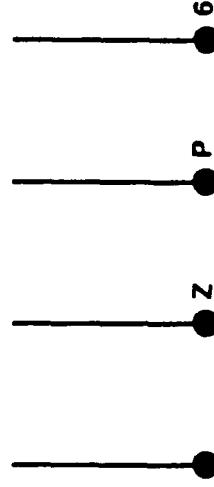
RELATIONSHIP LABELS

EMPLOYEE / 4

HAS

DEPENDENT / 5

SPECIFIC RELATIONSHIPS HAVE ONLY ONE LABEL. IT IS READ FROM THE PARENT ENTITY TOWARD THE CHILD ENTITY.

PART / 1

IS SUPPLIED BY/SUPPLIES

SUPPLIER /3

RELATIONSHIPS ARE LABELED WITH A VERB OR VERB PHRASE.

NON-SPECIFIC RELATIONSHIPS ARE LABELED IN EACH DIRECTION. THE TWO LABELS ARE SEPARATED BY A SLASH. THE LABEL TO THE LEFT OF THE SLASH IS READ WHEN READING THE RELATIONSHIP FROM TOP ENTITY TO BOTTOM ENTITY OR LEFT ENTITY TO RIGHT ENTITY.
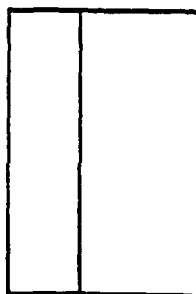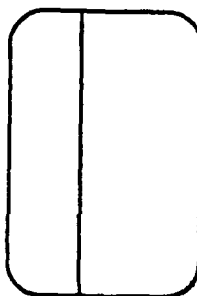
ALTERNATE KEYS

- SOMETIMES THERE MAY BE MORE THAN ONE ATTRIBUTE OR COMBINATION OF ATTRIBUTES THAT UNIQUELY IDENTIFIES AN INSTANCE OF AN ENTITY.

- WHEN THERE IS A CHOICE OF KEYS, ONE MUST BE CHOSEN AS THE PRIME KEY. THE OTHER(S) BECOME ALTERNATE KEYS.

- ALTERNATE KEYS ARE PLACED BELOW THE LINE IN THE ENTITY BOX AND IDENTIFIED WITH THE LETTERS AK IN PARENTHESES PLUS A NUMBER.

EXAMPLE:

EMPLOYEE / 1

Employee Number
Soc Sec Number(AK1) Emp Name(AK2) Emp Birthdate(AK2)

- ALTERNATE KEYS NEVER MIGRATE

324 10
101

KEY MIGRATION AND
IDENTIFIER DEPENDENCY

EMPLOYEE / 1

Employee No.

PARENT
ENTITY

EMPLOYEE / 1

Employee No.

has

AUTOMOBILE / 2

License No.
Employee No. (FK)

owns

CHILD
ENTITY

DEPENDENT / 3

Employee No. (FK) Dependent Name

IN THIS CASE, THE CHILD ENTITY
AUTOMOBILE HAS ITS OWN
UNIQUE IDENTIFIER, THE
LICENSE NUMBER. THE KEY OF
THE PARENT MIGRATES AS A
NON-KEY ATTRIBUTE. THE
RELATIONSHIP LINE IS
THEREFORE A DASHED LINE.

SINCE MORE THAN ONE
EMPLOYEE MAY HAVE A CHILD
WITH THE SAME NAME, THE
KEY OF THE PARENT ENTITY,
EMPLOYEE, IS REQUIRED TO
MIGRATE TO FORM PART OF
THE KEY OF THE CHILD ENTITY,
DEPENDENT. THEREFORE THE
RELATIONSHIP LINE IS SOLID.

WHEN THE CHILD DEPENDS
UPON THE KEY OF THE PARENT
FOR ITS IDENTITY, THE ENTITY
BOX IS MODIFIED TO HAVE
ROUND OR BEVELED
CORNERS.

324.11
1101

FOREIGN KEY ROLE NAMES

- WHEN A KEY MIGRATES TWICE INTO AN ENTITY THERE IS AN APPARENT VIOLATION OF THE RULE FOR ATTRIBUTES IN AN ENTITY THAT EACH ATTRIBUTE IS UNIQUE WITHIN THE ENTITY.

- WHEN BOTH MIGRATED KEYS ARE NECESSARY, THIS DOUBLE APPEARANCE CAN BE RESOLVED BY GIVING EACH A "ROLE NAME" WHICH IDENTIFIES THE MEANING OF EACH IN THE CONTEXT OF THE CHILD ENTITY

- THE FORMAT FOR A ROLE NAME IS "ROLE NAME.FOREIGN KEY(FK)" WITH THE ROLE NAME SEPARATED FROM THE FOREIGN KEY BY A PERIOD

ROLE NAME EXAMPLE

THE ROLE NAMES "SUPERIOR"
AND "SUBORDINATE" GIVE
MEANINGS TO THE TWO
EMPLOYEE NUMBER MIGRATED
KEYS IN THE CONTEXT OF THE
REPORTING STRUCTURE ENTITY
AND RESULT IN UNIQUE NAMES
FOR EACH OF THE TWO
ATTRIBUTES IN THE KEY.

EMPLOYEE / 1

Employee No.

is superior in

is subordinate in

REPORTING STRUCTURE / 4

Superior.Employee No. (FK)
Subordinate.Employee No. (FK)

THESE TWO RELATIONSHIPS
BETWEEN TWO ENTITIES
REPRESENT THE RESOLUTION OF
A MANY-TO-MANY RELATIONSHIP
BETWEEN INSTANCES OF THE
PARENT ENTITY RATHER THAN
INSTANCES OF TWO DIFFERENT
ENTITIES. THE ZERO, ONE OR
MANY CARDINALITIES REPRESENT
A NETWORK. THIS MODEL STATES
THAT AN EMPLOYEE CAN BE
SUBORDINATE TO MORE THAN ONE
OTHER EMPLOYEE. THAT IS, AN
EMPLOYEE CAN HAVE MORE THAN
ONE BOSS.

IF THE CARDINALITY ON THE
SUBORDINATE RELATIONSHIP
WERE ZERO OR ONE, THEN THE
HIERARCHICAL RELATIONSHIP
WOULD EXIST AND AN EMPLOYEE
COULD ONLY HAVE ONE
SUPERIOR.

GENERALIZATION
(OR CATEGORIZATION)

SINCE THE SUB-TYPE IS THE SAME THING AS THE GENERALIZATION ENTITY, THE KEY OF THE SUB-TYPE IS EXACTLY THE SAME AS THE GENERALIZATION ENTITY.

GENERALIZATION
(OR CATEGORY)
ENTITY

THE DISCRIMINATOR IS AN ATTRIBUTE OF THE GENERALIZATION ENTITY

EMPLOYEE / 1

Employee Number
SSN(AK1)
Emp Name
Birthplace

PAYROLL

SEX

SUB-TYPES

HOURLY EMP / 2

Employee Number(FK)

SALARIED EMP / 3

Employee Number(FK)

MALE EMP / 4

Employee Number(FK)

FEMALE EMP / 5

Employee Number(FK)

THE SINGLE LINE BELOW THE DISCRIMINATOR MEANS THAT NOT ALL POSSIBLE SUB-TYPES ARE SHOWN. FOR AN INSTANCE OF THE GENERALIZATION ENTITY THERE IS NOT NECESSARILY AN INSTANCE OF A SUB-TYPE.

THE DOUBLE LINE BELOW THE DISCRIMINATOR MEANS THAT THE COMPLETE SET OF SUB-TYPES IS SHOWN. FOR EACH INSTANCE OF THE GENERALIZATION ENTITY THERE MUST BE EXACTLY ONE INSTANCE OF A SUB-TYPE

ELIMINATION OF TRANSITIVE DEPENDENCIES

MACHINE / 6

Machine No.	
	Machine Type Maintenance Interval Date of Last Maint.

MACHINE CLASS / 9

Machine Type	
	Maintenance Interval

- - - ○< contains ●
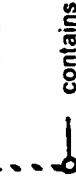
MACHINE / 6

Machine No.	
	Machine Type(FK) Date of Last Maint.

THERE MAY BE A NUMBER OF MACHINES OF THE SAME TYPE. THE MAINTENANCE INTERVAL IS THE SAME FOR ALL MACHINES OF THE SAME TYPE.

THE MAINTENANCE INTERVAL DOES NOT DEPEND UPON THE KEY, "MACHINE NO." BUT UPON A NON-KEY ATTRIBUTE, "MACHINE TYPE".

THIS "TRANSITIVE DEPENDENCY" IS CORRECTED AT THE RIGHT.

324 15
1911

NOTICE

DACOM
(c) D. Appleton Co. Inc

IDEF1X IS AN EXTENDED

VERSION OF IDEF1

- IS UPWARD COMPATIBLE WITH THE PRIOR VERSION

- PROVIDES IMPROVED DEFINITION OF DATA REQUIREMENTS

IDEF1X 1A

MAJOR CHANGES :

- SYNTAX

- CATEGORIZATION

- NOTES

- "PUFFY" BOXES

- ENTITY IDENTIFICATION

- FOREIGN KEY IDENTIFICATION

- ATTRIBUTE ROLE NAMES

IDEF1X 2

ENTITY CONSTRUCT

IDEF1X

IDENTIFIER-INDEPENDENT ENTITY

EMPLOYER/31
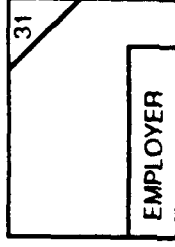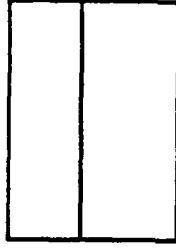
IDENTIFIER-DEPENDENT ENTITY

EMPLOYEE/32

IDEF1

ENTITY

31

EMPLOYER

Not Designated

IDEF1X 3

RELATIONSHIPS

IDEF1X

IDENTIFIER
DEPENDENCE

————————

Dependent on
Parent Entity
for Identification

٠٠٠٠٠٠٠

Independent from
Parent Entity
for Identification

IDEF1

Implied by Underscore
on Foreign-Key
attributes (inherited
attributes)

IDEF1X 4

EXAMPLES

IDEF1

IDEF1X

IDEF1X 5

EXAMPLES

IDEF1

A
ALPHA
1

—◁— RELATES TO

B
A
W
X
BETA
2

IDEF1X

ALPHA/1
A

—○— RELATES TO
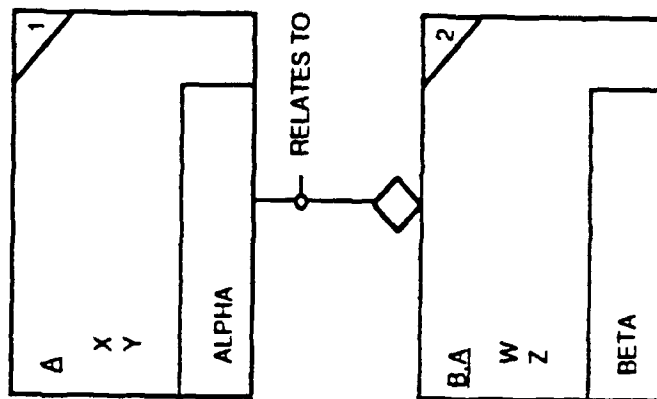
BETA/2
B
A(FK)
W
X
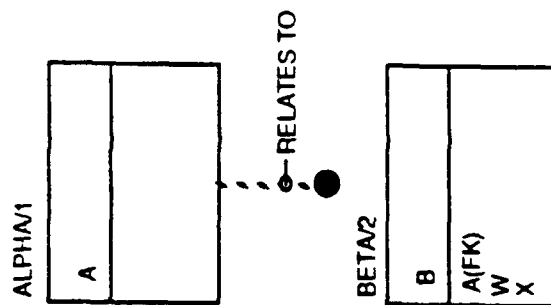
B IS IDENTIFICATION INDEPENDENT
FROM A

IDEF1X 6

EXAMPLES

IDEF1X

ALPHA/2

A

BETA/3

B

Relates to

Relates to

ALPHA/2 ... TAU/5

T
A(FK)
B(FK)
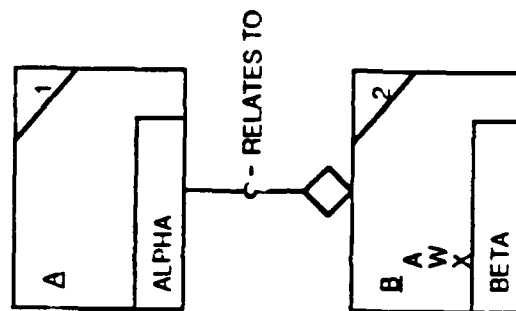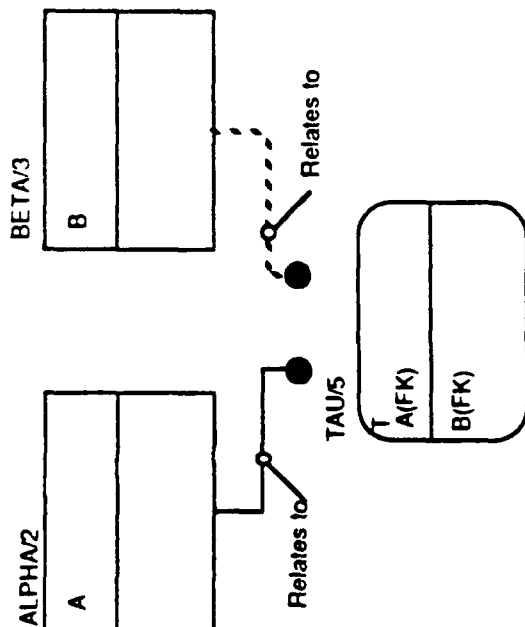
TAU IS IDENTIFIER DEPENDENT UPON A
AND INDEPENDENT FROM B

IDEF1

A   2

ALPHA

B   3

BETA

Relates to

Relates to

A1   4

B

TAU
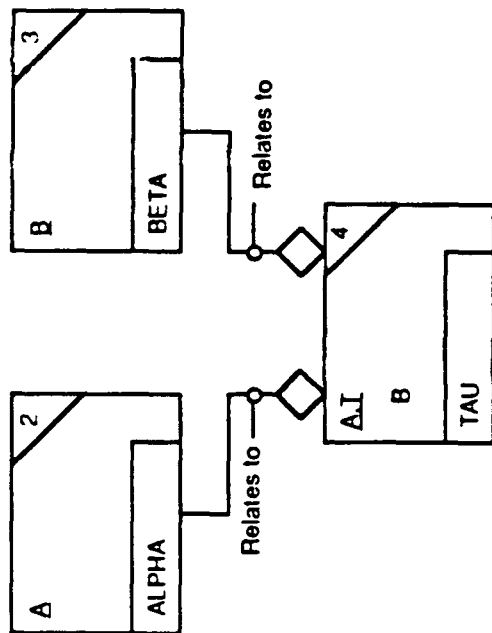
ATTRIBUTES

IDEF1X

IDEF1

PRIMARY KEY

XXX
YYY
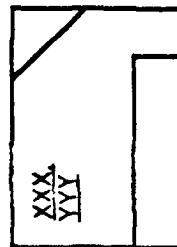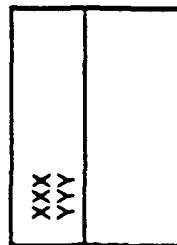
XXX
YYY
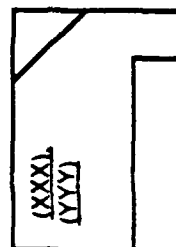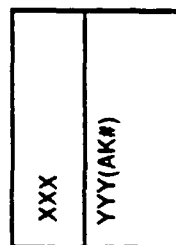
ALTERNATE KEY
ATTRIBUTES

XXX
YYY(AK#)

(XXX)
YYY

FOREIGN KEY
ATTRIBUTES

XXX (FK)
YYY (FK)
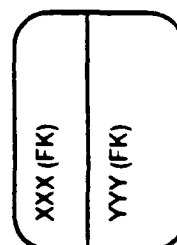
Not Designated

RELATIONSHIPS

IDEF1

IDEF1X

ASSOCIATION
(i.e. Aggregation relationship)

CARDINALITY

Graphic
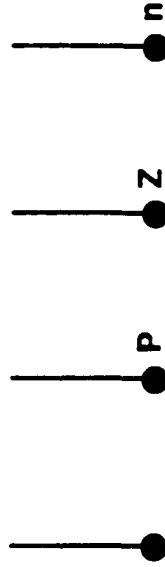
Meaning
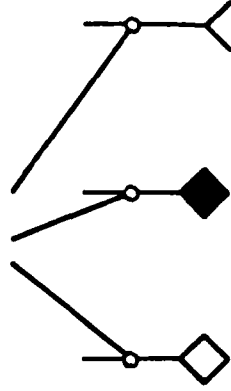
IDEF1X 9

EXAMPLES

IDEF1

SPACE NO | 10

PKNG SPACE

is

EMP NO | 6

EMPLOYEE

has

EMP NO
SPACE NO | 11

ASSGND SPC

IDEF1X

PARKING SPACE/10

SPACE NO

is

Z

ASSIGNED SPACE/11

EMP NO(FK)
SPACE NO(FK)

EMPLOYEE/6

EMP NO

Z

has
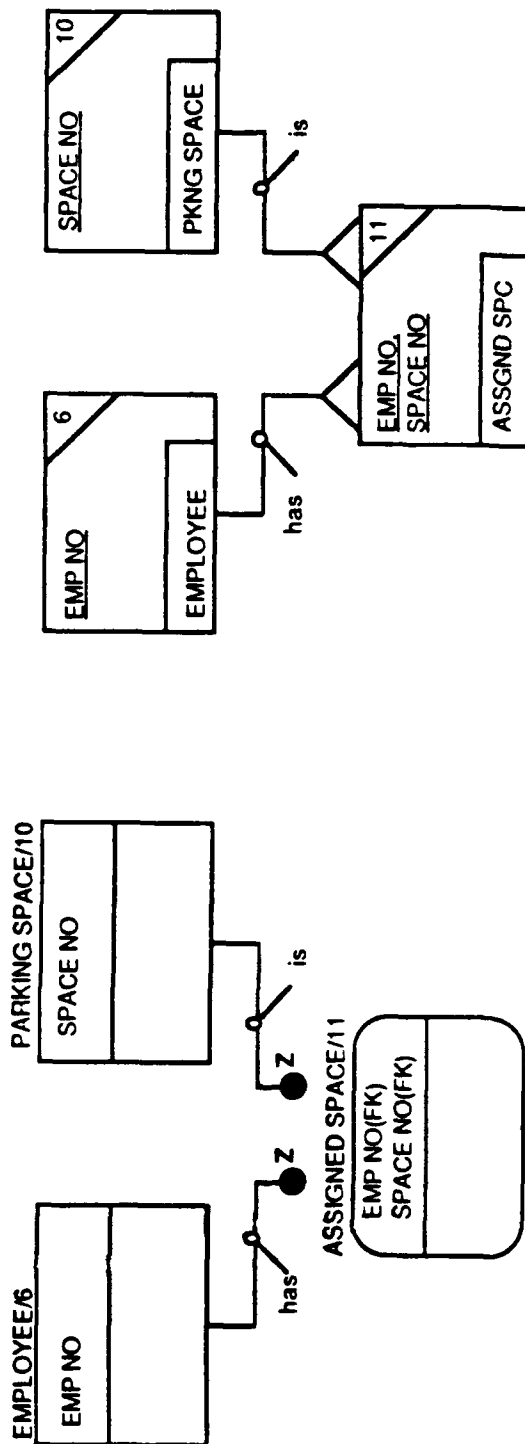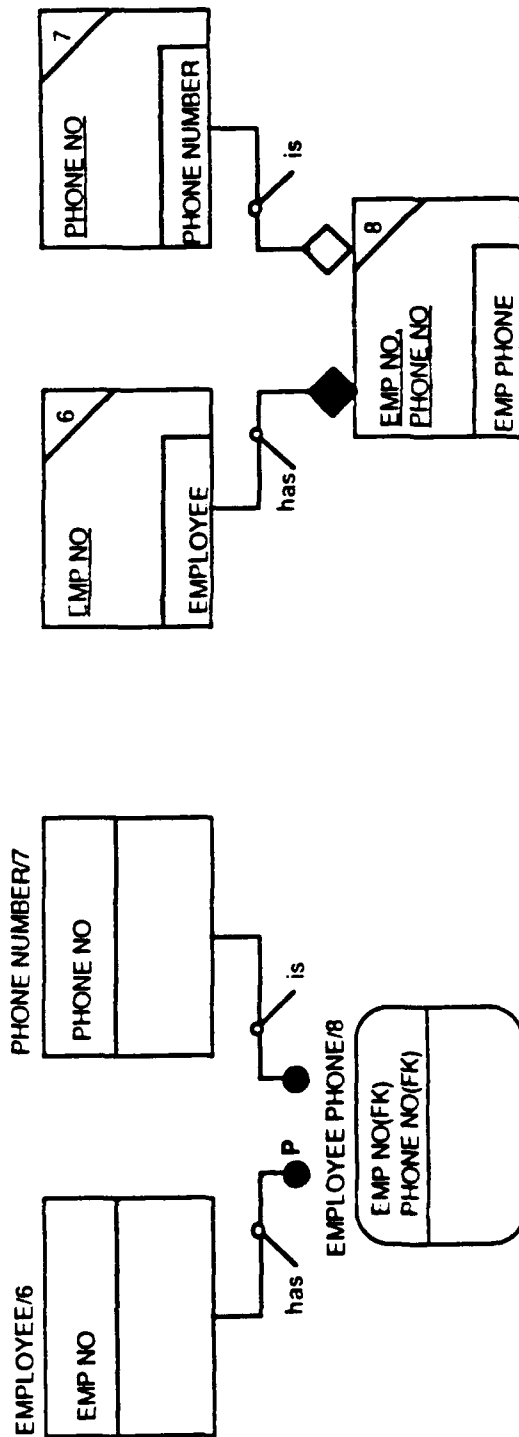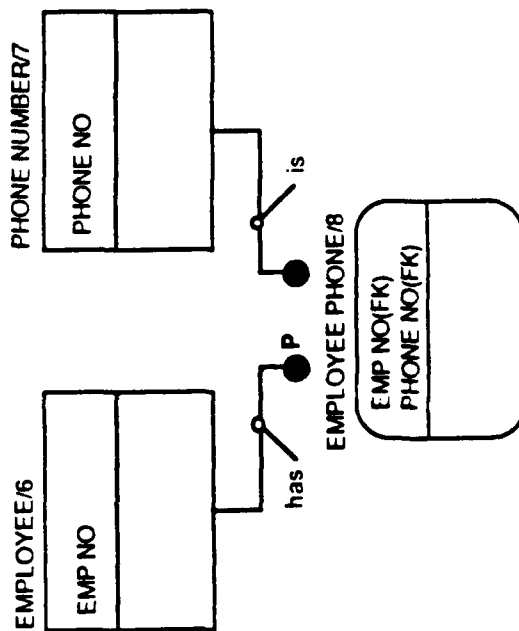
IDEF1X-10

EXAMPLES

IDEF1
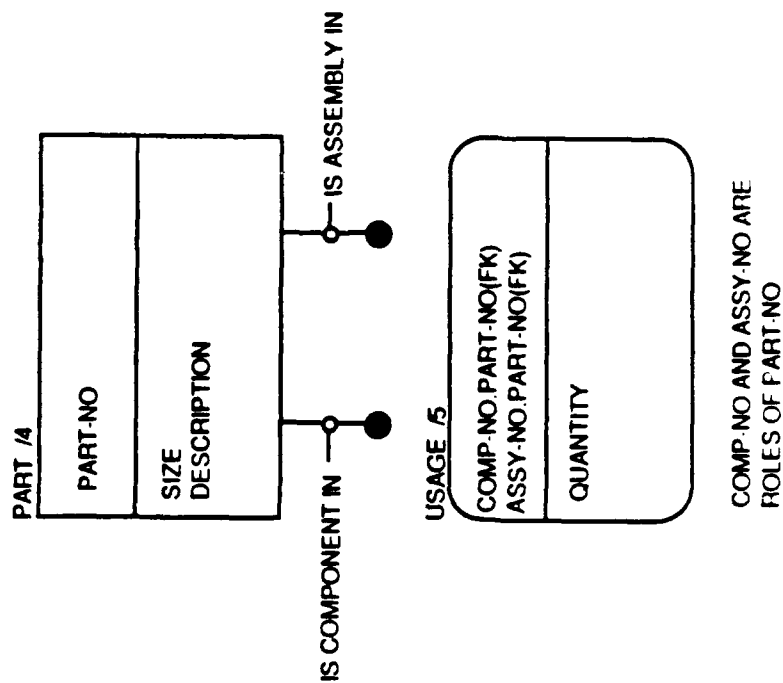

IDEF1X

ATTRIBUTES

IDEF1X

IDEF1

ATTRIBUTE WITH
ROLE NAME

ROLE NAME.PRIMARY NAME

e.g. Component No.Part No(FK)

Not Designated
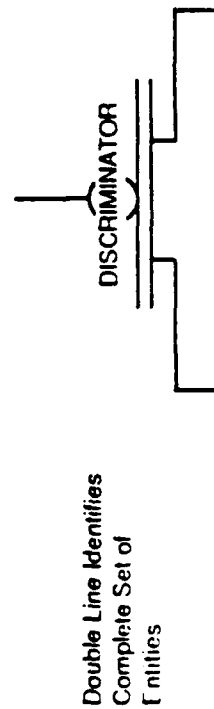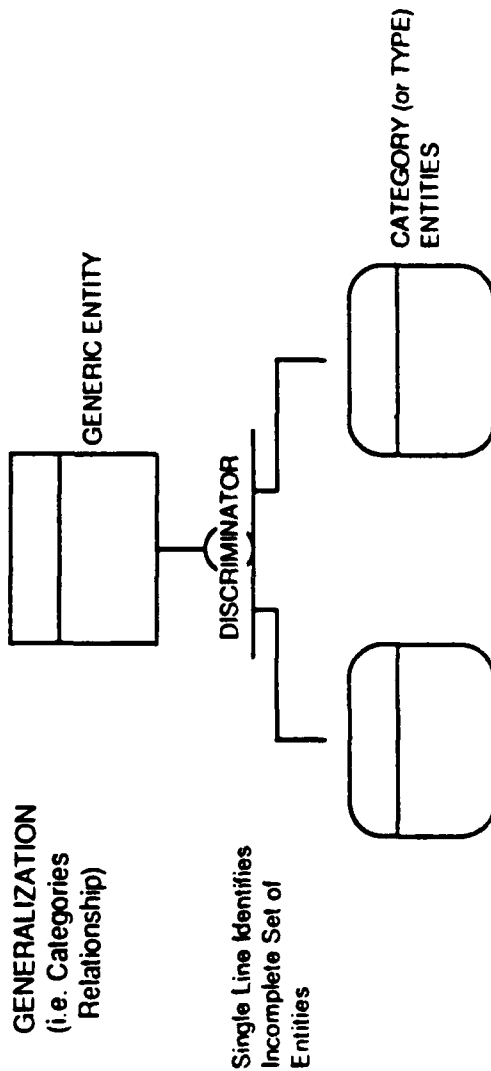
IDEF1X-12

EXAMPLES

IDEF1

IDEF1

4

PART-NO
SIZE
DESCRIPTION

PART

IS ASSEMBLY IN

IS COMPONENT IN

5

PART-NO,PART-NO
QUANTITY

USAGE

IDEF1X

IDEF1X

PART /4

PART-NO

SIZE
DESCRIPTION

IS ASSEMBLY IN

IS COMPONENT IN

USAGE /5

COMP-NO.PART-NO(FK)
ASSY-NO.PART-NO(FK)

QUANTITY

COMP-NO AND ASSY-NO ARE
ROLES OF PART-NO

IDEF1X 13

RELATIONSHIPS

IDEF1X

IDEF1

GENERIC ENTITY

NOT DESIGNATED

CATEGORY (or TYPE)
ENTITIES

DISCRIMINATOR

DISCRIMINATOR

GENERALIZATION
(i.e. Categories
Relationship)

Single Line Identifies
Incomplete Set of
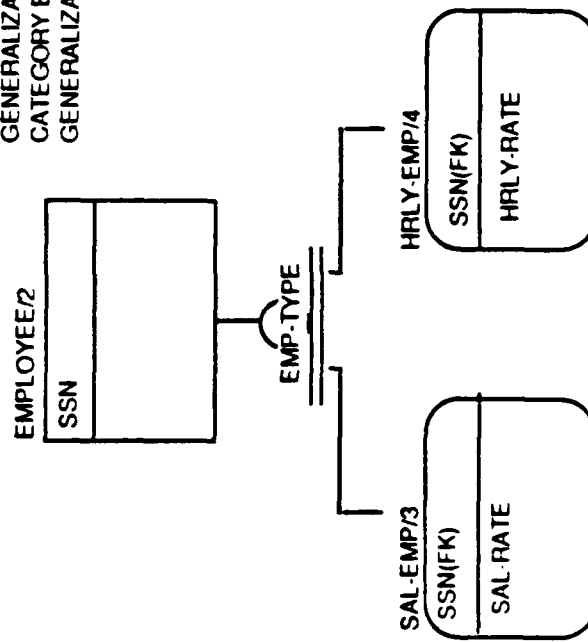Entities

Double Line Identifies
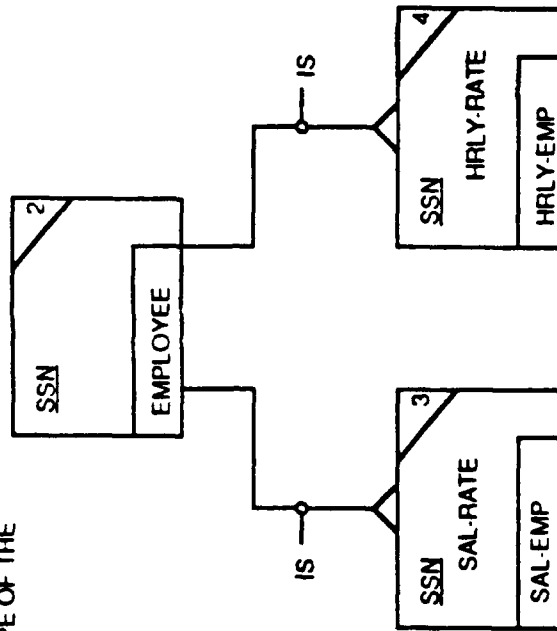Complete Set of
Entities

IDEF1X 14

EXAMPLES

IDEF1

IDEF1X

NOTE: THE KEY OF EACH CATEGORY ENTITY IS EXACTLY THE SAME AS THE KEY OF THE GENERALIZATION ENTITY BECAUSE THE CATEGORY ENTITY IS A SUB-TYPE OF THE GENERALIZATION TYPE.

Appendix D2

Nijssen Information Analysis Method (NIAM)

NOTICE

NIAM

Information Model

Paul Thompson

Control Data

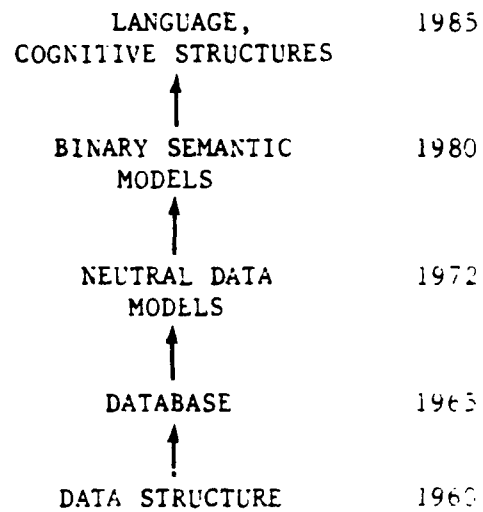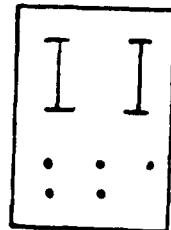86.01.01

CONTROL DATA

WHAT IS NIAM?

1. A binary semantic conceptual model.

2. A means of capturing (complex) information requirements in user understandable terms,

    modeling and analyzing the requirements in a feature-rich formal information model,

    and translating conceptual information requirements into implementable specifications.

WHY NIAM?

o   *More power and precision in knowledge representation*

o   More discipline and rigor in the methodology

o   Greater user (expert) participation

o   Cleaner separation of expert models from specification models

o   Easier generation of neutral data model specifications

o   More real-world semantics for data and rule bases

User-Machine Gap

LANGUAGE, COGNITIVE STRUCTURES	1985
↑	
BINARY SEMANTIC MODELS	1980
↑	
NEUTRAL DATA MODELS	1972
↑	
DATABASE	1965
↑	
DATA STRUCTURE	1960
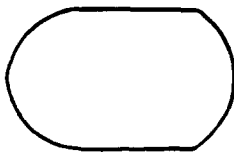
BACKGROUND

ca.1970   Recognition of need for better data modeling
          techniques

1972   NIAM research into information analysis

1975   ANSI/X3/SPARC − 3 Schema Architecture
          Conceptual Schema proposal

1975   NIAM real−world applications

1980   NIAM released

1981   IFIP−CRIS conferences
          NIAM rated high

1982   ISO TC97/SC5/WG3
          Evaluation of Conceptual Schemas proposals
          NIAM selected as binary approach

1985   NIAM used within PDES initiation effort

1986   NIAM usage worldwide
          > 120 organizations
          > 5 universities

Information Analysis

CD CONTROL DATA

The *REAL WORLD* or
Environment is Described by
Concepts
Ideas
Constraints
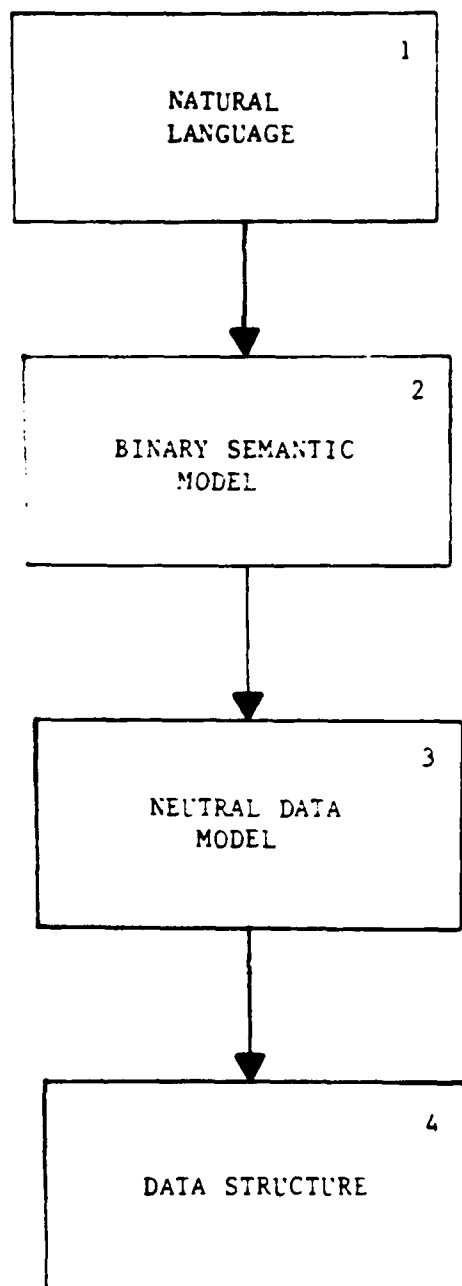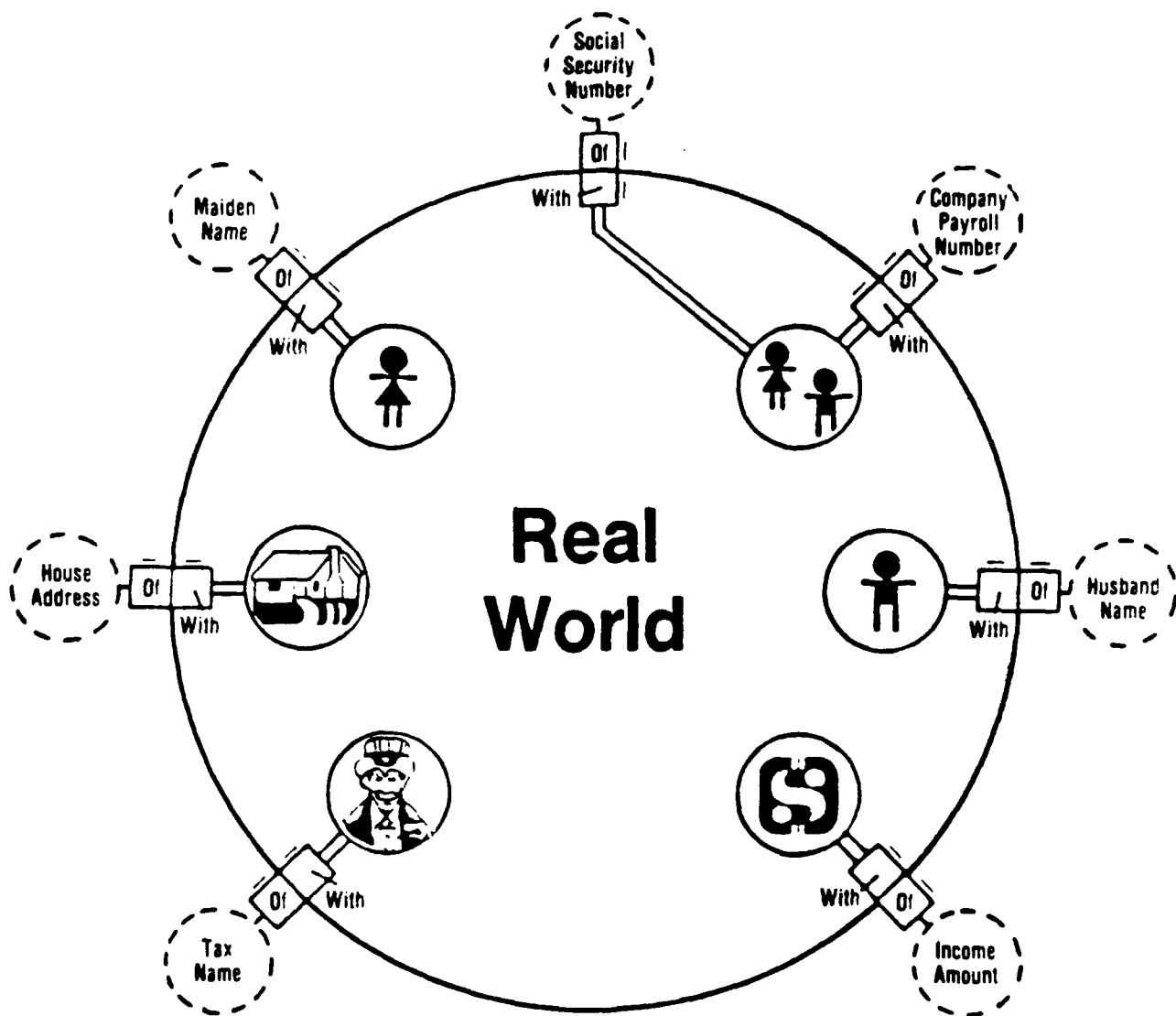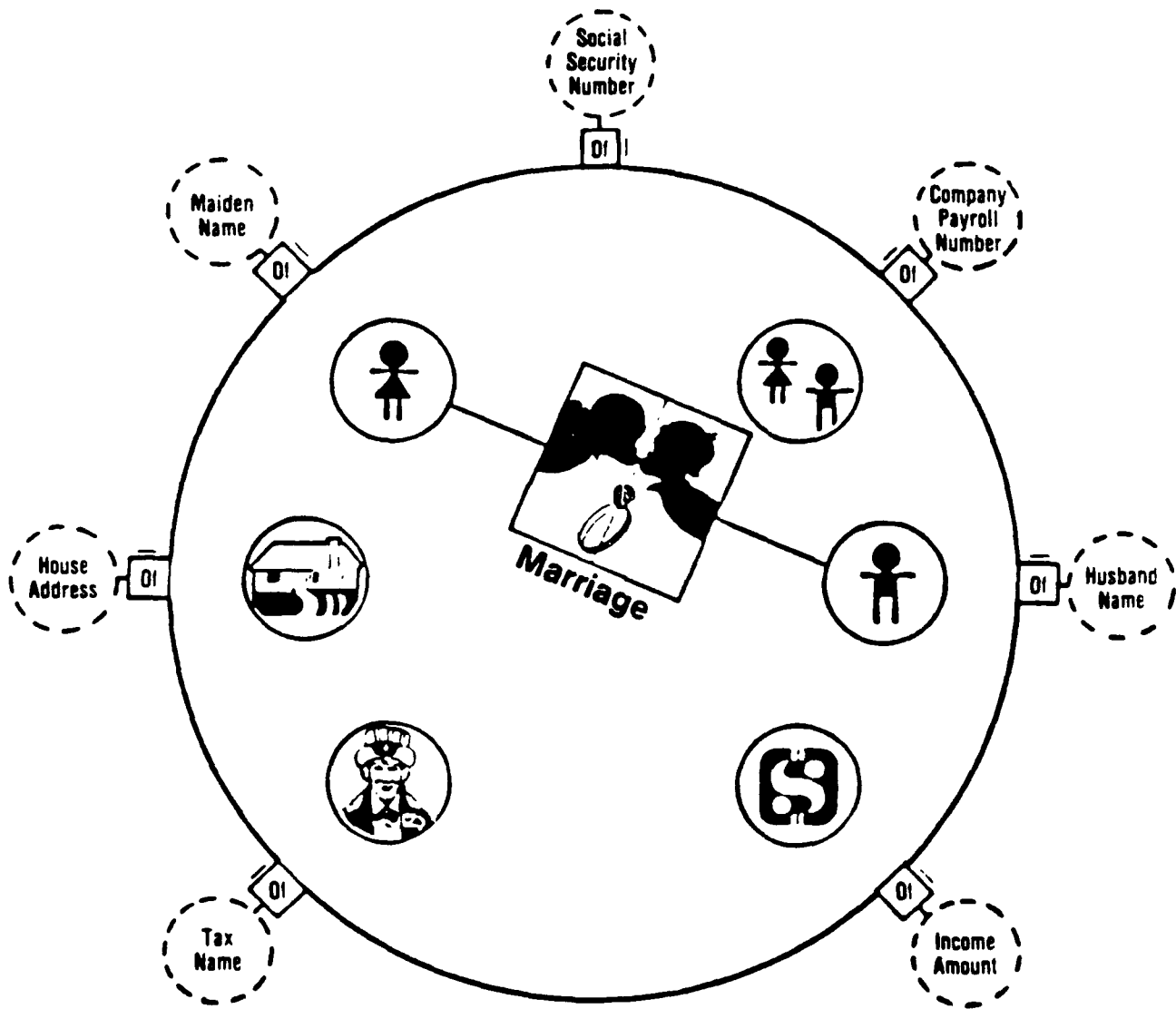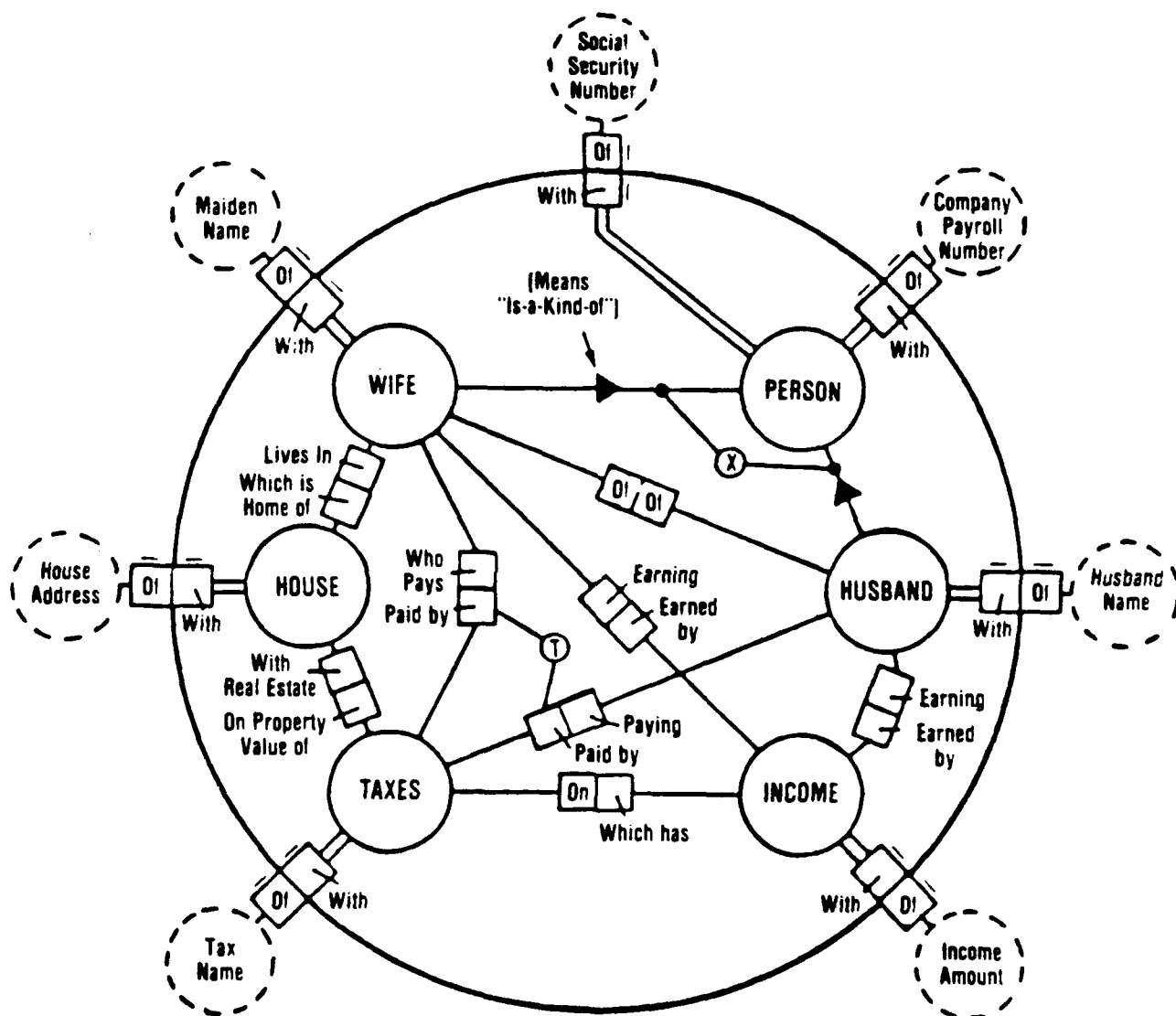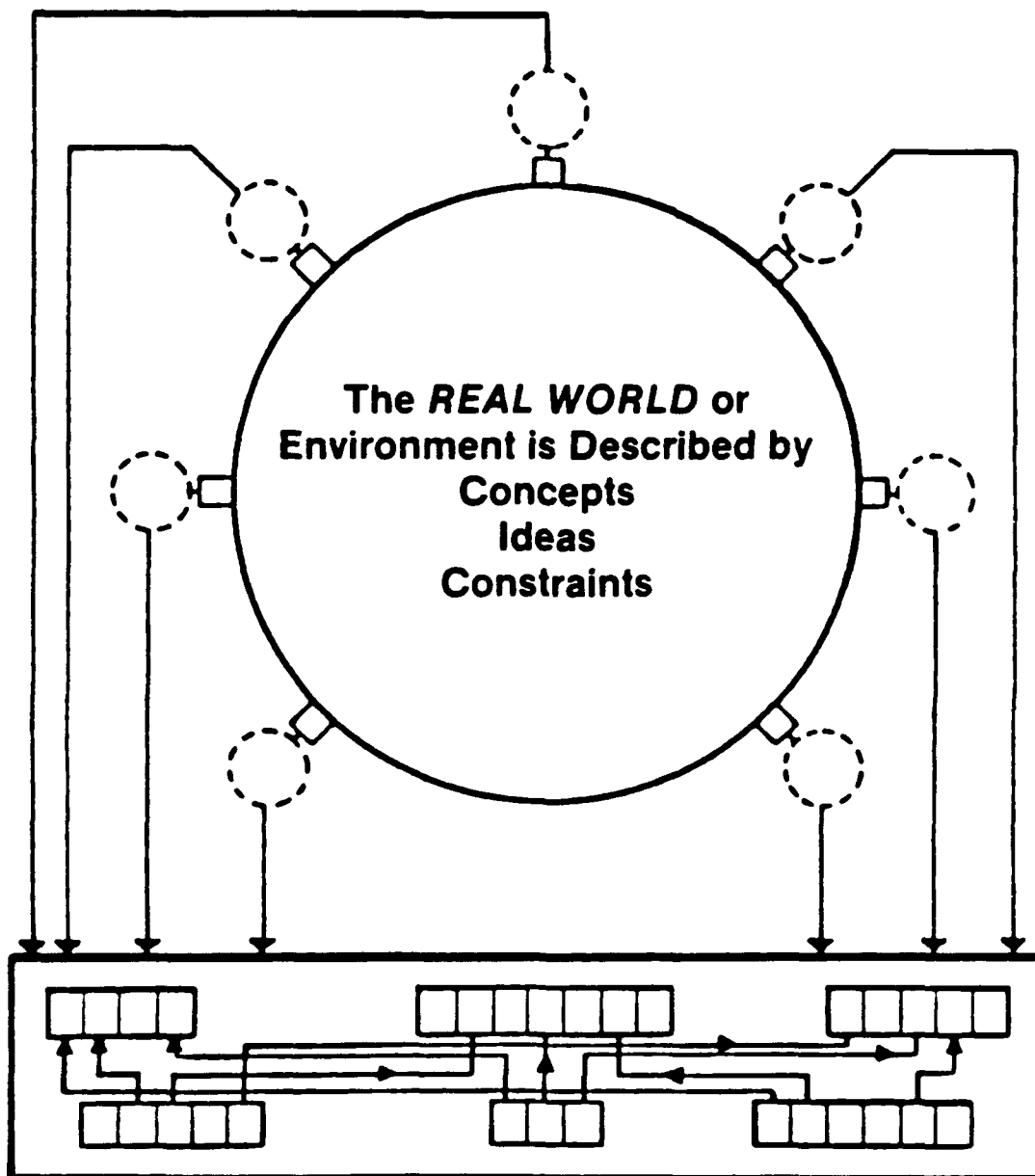
**The Information System Contains a Lexical or Coded
Character-based Data-model**

Concept Type

A concept type is a set of instances within a Universal of Discourse having common properties.
A concept type is independent from symbols used to represent it.
A subtype shares (inherits) all properties of its supertype. A subtype may have multiple supertypes.
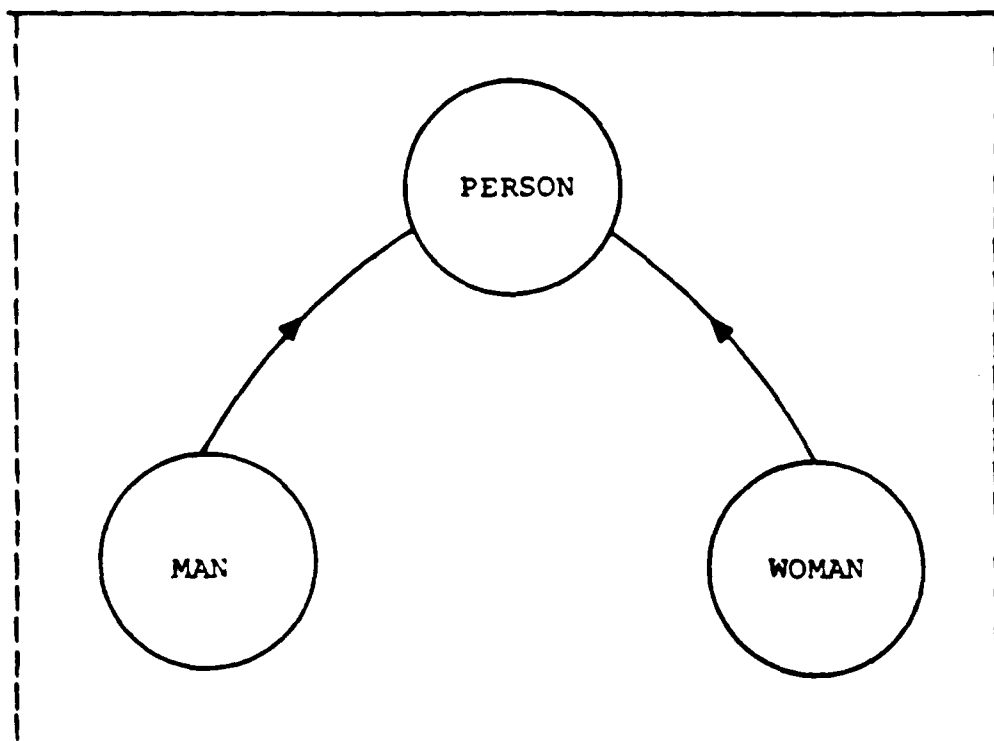
Total Subtype Constraint

A total subtype constraint prescribes that for every occurrence of a certain concept type there must be an occurrence of one of the specified subtypes.
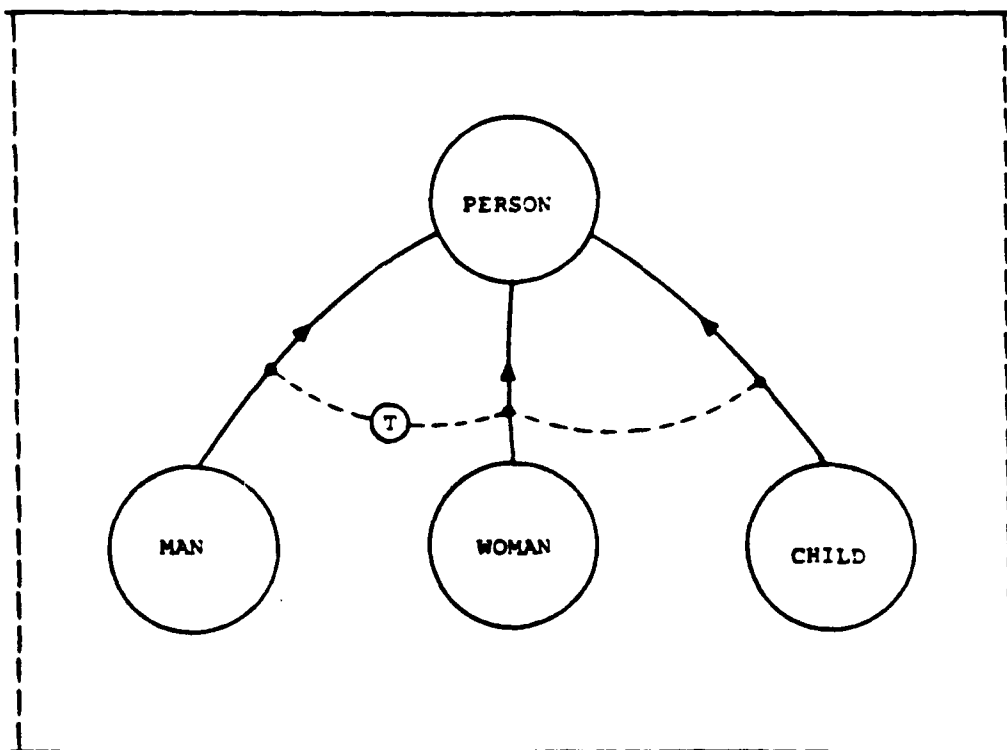
Subtype Exclusion

A subtype exclusion constraint mutually excludes the sets of occurrences of
two or more subtypes within one concept type family.
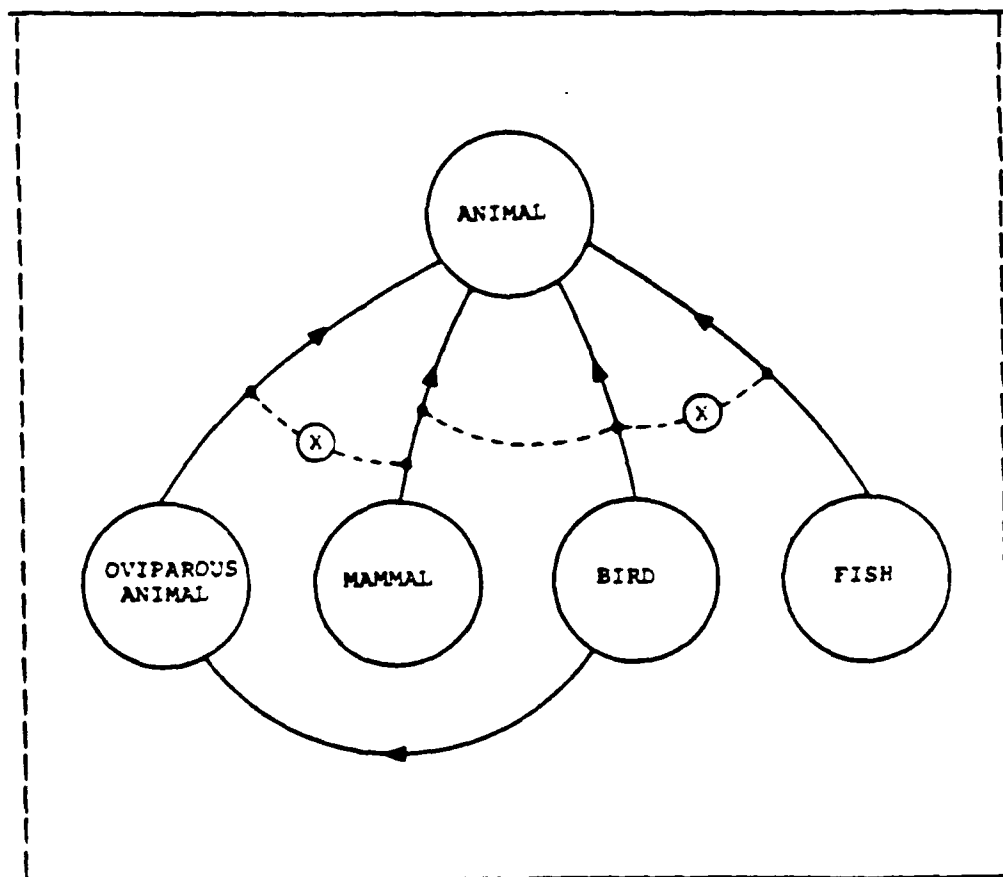
Idea Type

An idea type is an information-bearing connection of concept types.
An idea type has two different roles, each role being played by one concept
type.
An idea type can have uniqueness constraint(s) indicating the identifying
role(s).
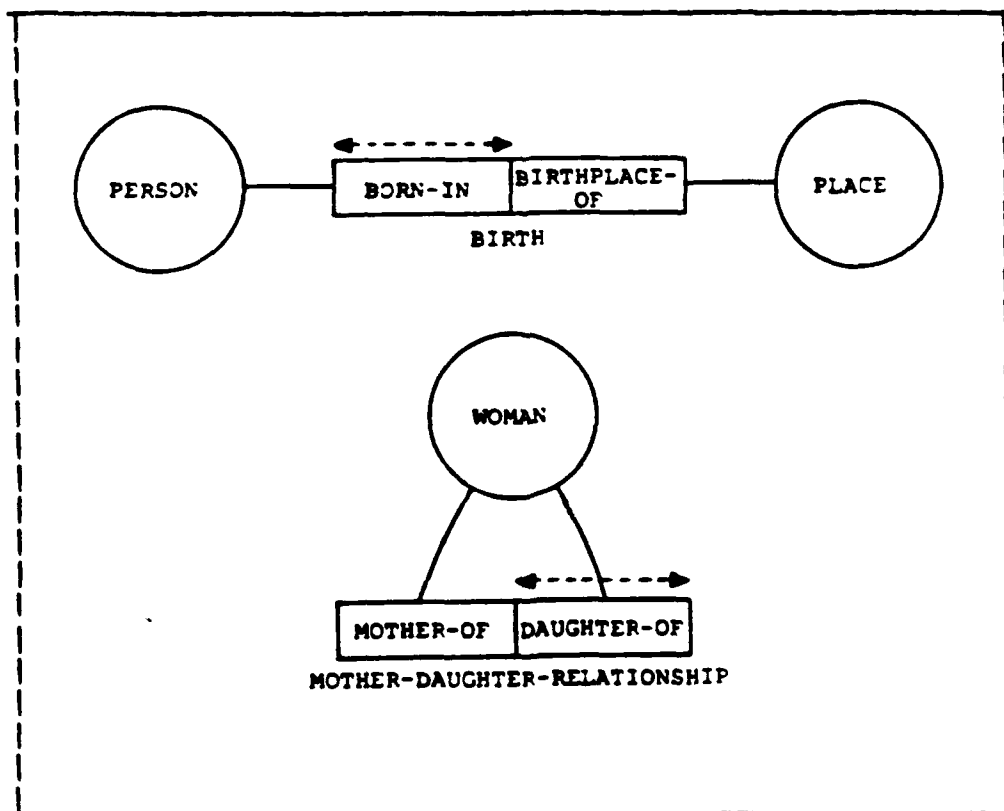
Uniqueness Constraint

A uniqueness constraint restricts the instances of one or more roles.
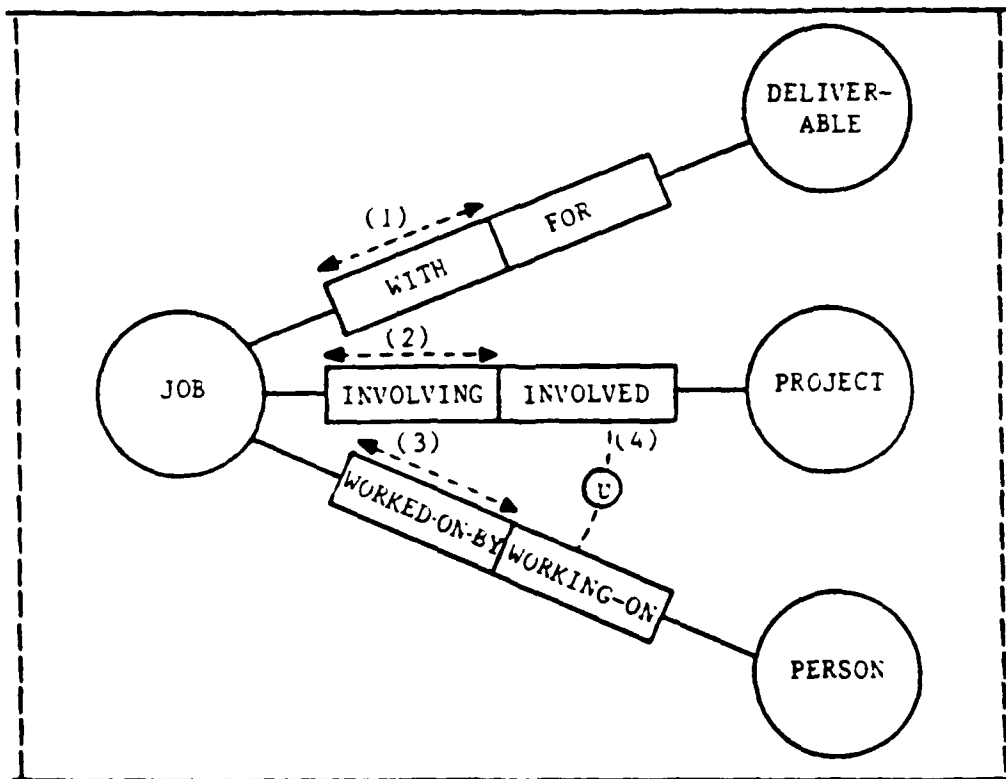
Role Subset Constraint

A subset constraint prescribes that the set of occurrences of one role (or role pair) is a subset of the set of occurrences of another role (or role pair).
According to the drawing conventions, the arrow points from domain to range (indicates what implies what).
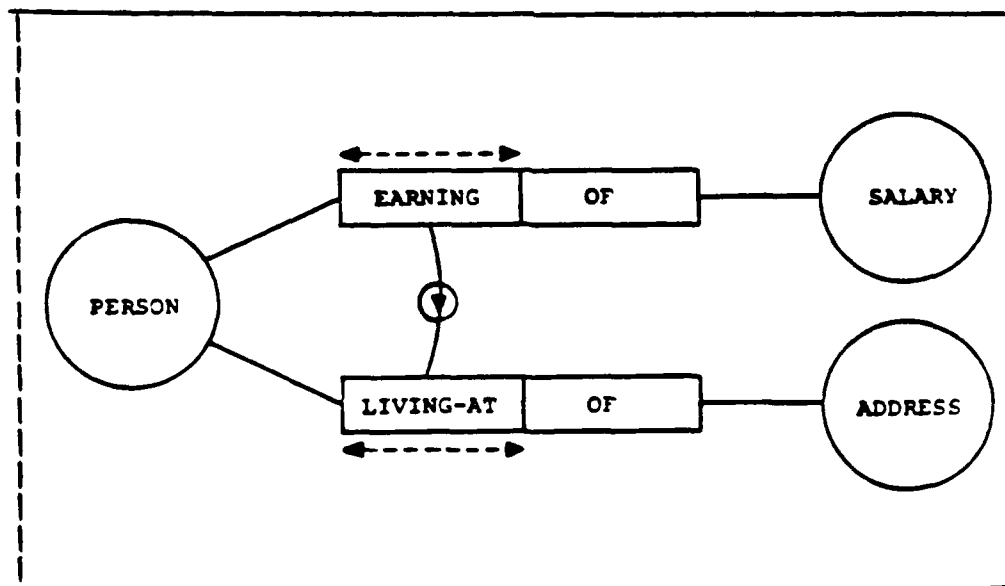
Cardinalities

A minimum and maximum cardinality may be expressed for each concept type and role.


```
   ┌──────────────────────────────────────────────────────────┐
   │                                                          │
   │                                                          │
   │    ╭──────────╮   ┌────────┬──────────┐   ╭──────────╮   │
   │    │          │   │        │BELONGING │   │          │   │
   │   ( COMMITTEE )──│  WITH  │   TO     │──( MEMBER   )  │
   │    │          │   └────────┴──────────┘   │          │   │
   │    ╰──────────╯      0,20       1,4       ╰──────────╯   │
   │        1,8                                   8,500       │
   │                                                          │
   └──────────────────────────────────────────────────────────┘
```
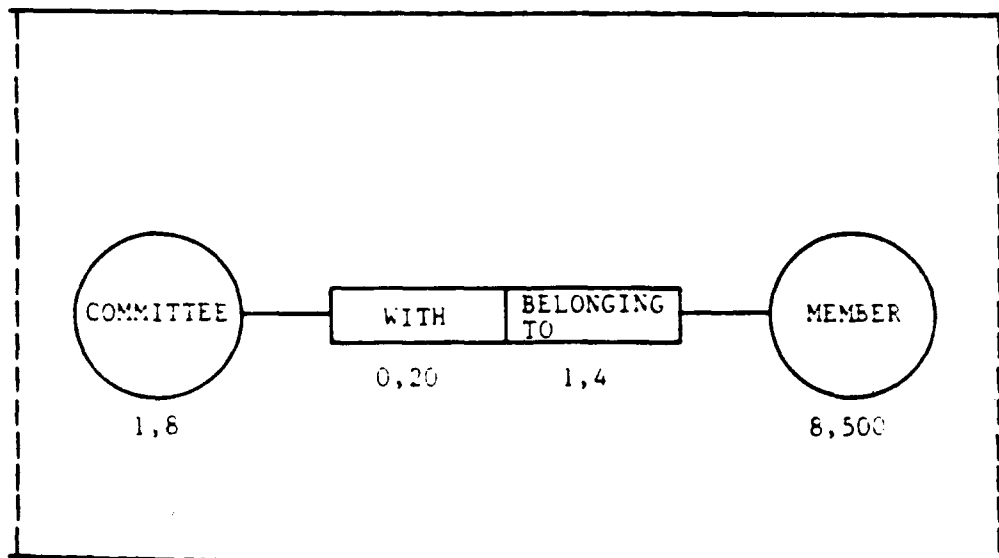
Total Role Constraint

A total role constraint prescribes that, for every occurrence of a concept, there must be an occurrence of a specified role.
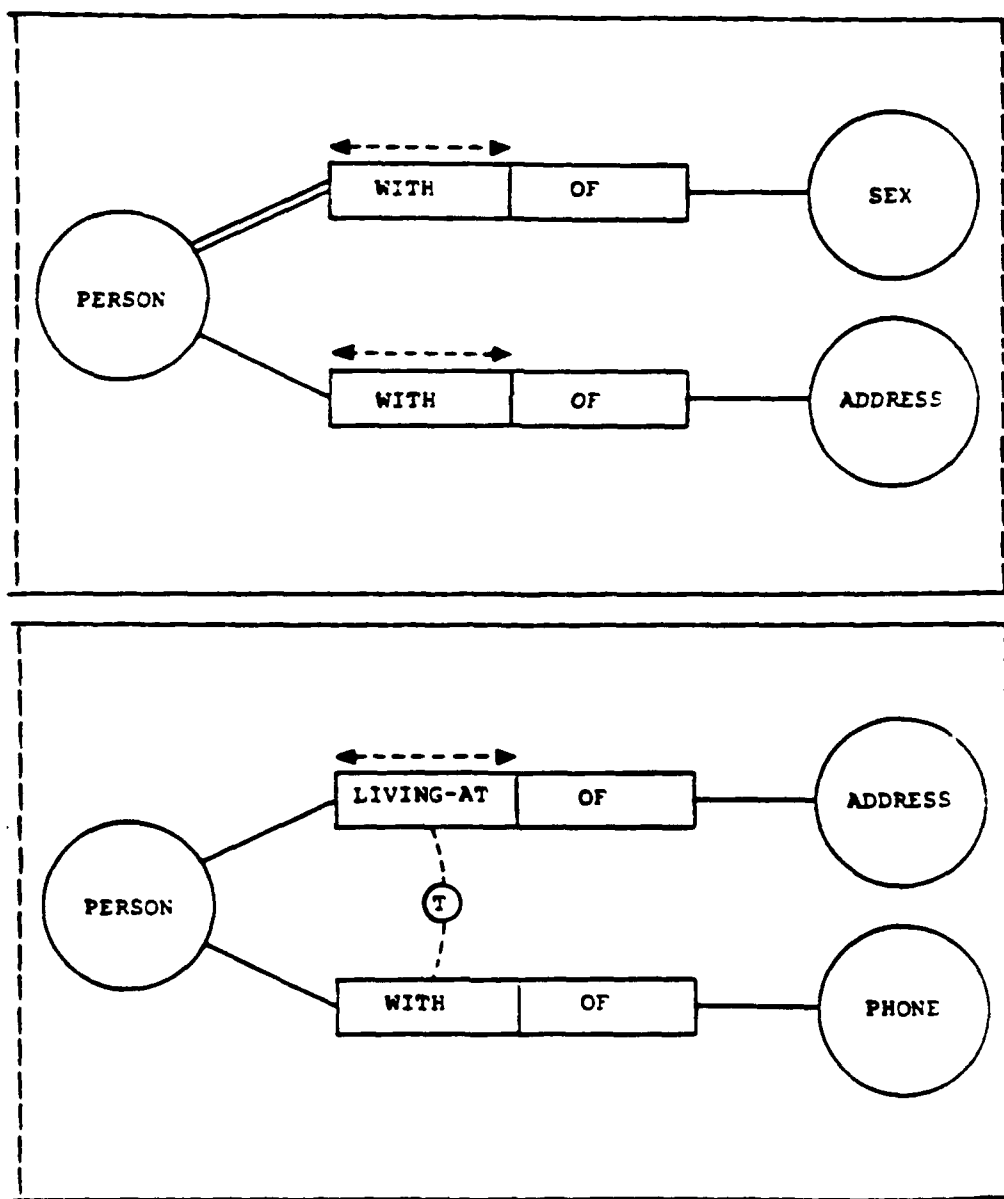
Role Exclusion Constraint

A role exclusion constraint prescribes that the set of occurrences of two roles (or role pairs) must be disjoint.
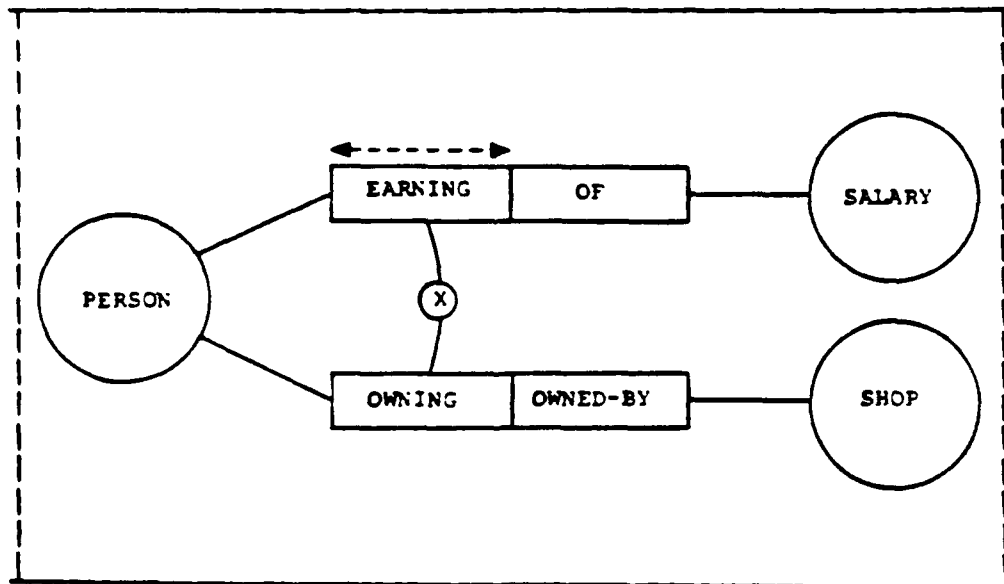
Equality Constraint

An equality constraint prescribes that the set of occurrences of two roles must be equal.
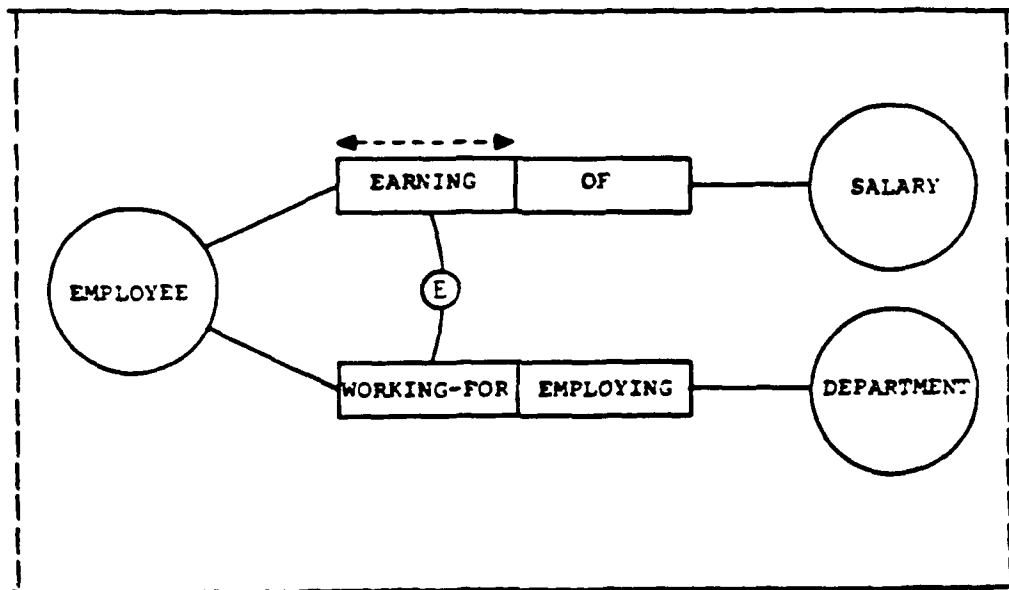
Combined Constraints

A person may have a primary skill and one or more secondary skills. Before having a primary skill the person must have a secondary skill. No secondary skill may be the duplicate of or equal to the primary.
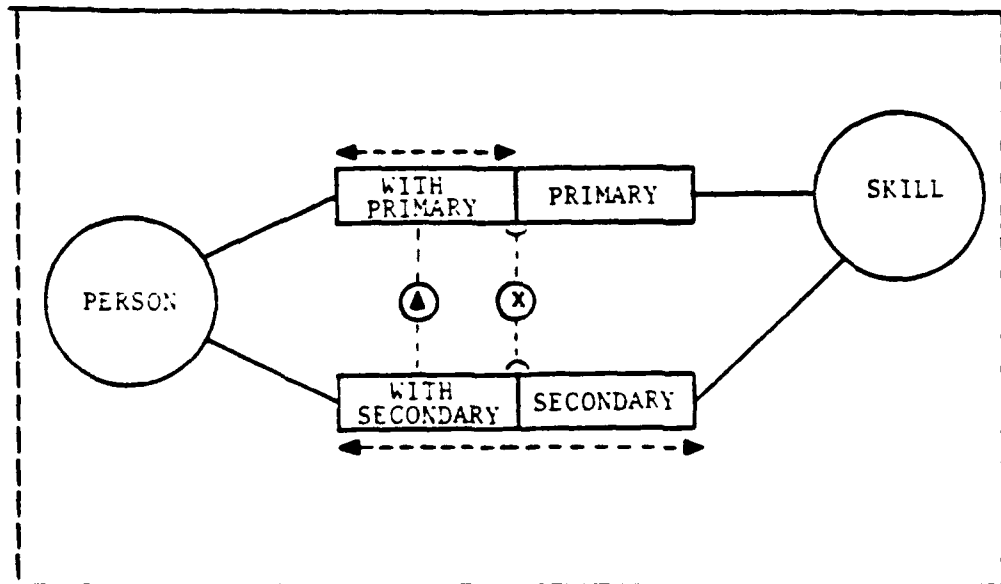
Advanced Constraints

Since each binary is a logical predicate, the power of predicate logic is available in a natural way for the expression of complex constraints.
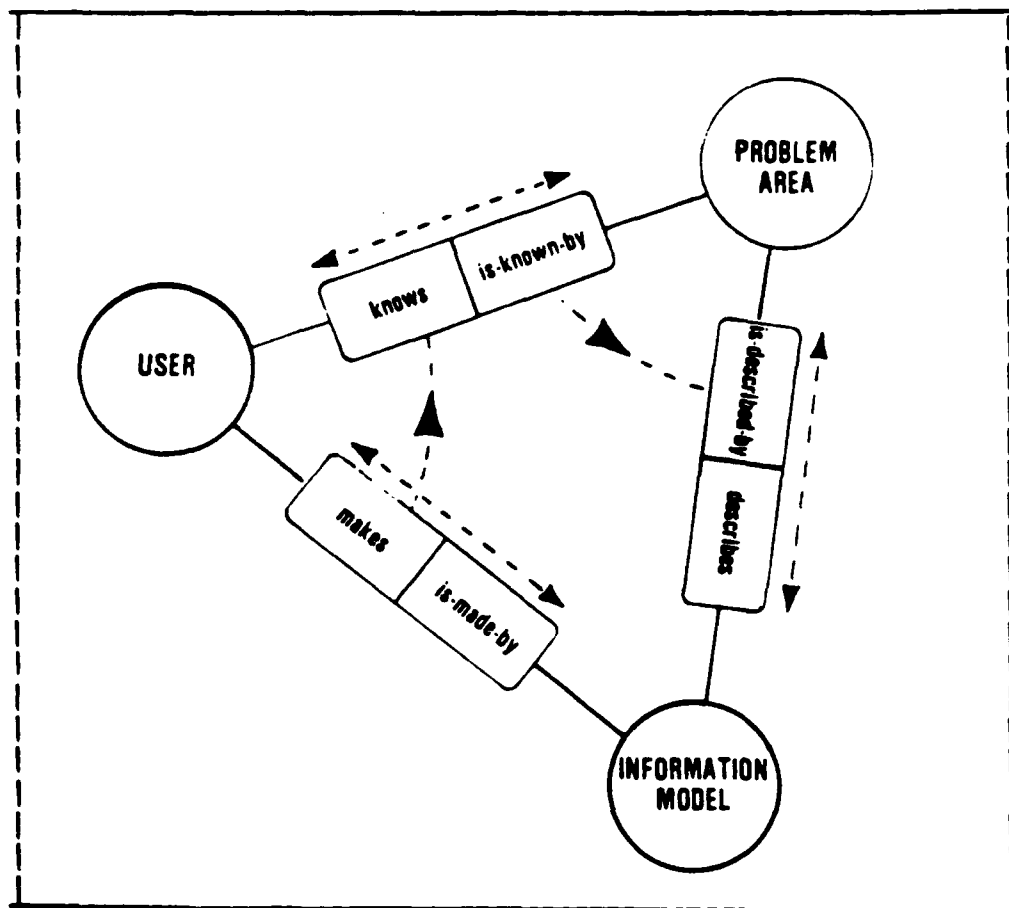
for each u: USER
If u makes an INFORMATION MODEL describing a PROBLEM AREA
then u must know the PROBLEM AREA.

Symbol Type

A symbol type is representable. It can be used to refer to a concept type (also called lexical object type).
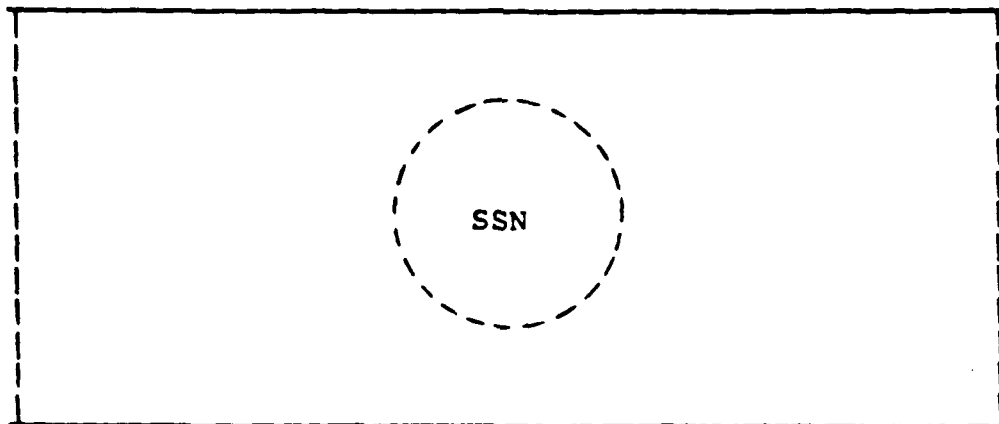
Bridge Type

A bridge type is the name-giving association between a concept type and a symbol type.
Depending on the position of uniqueness constraints on the bridge type, four different sorts of bridge types are recognized:

- One-to-one
- Synonym
- Homonym
- Syno-homonym
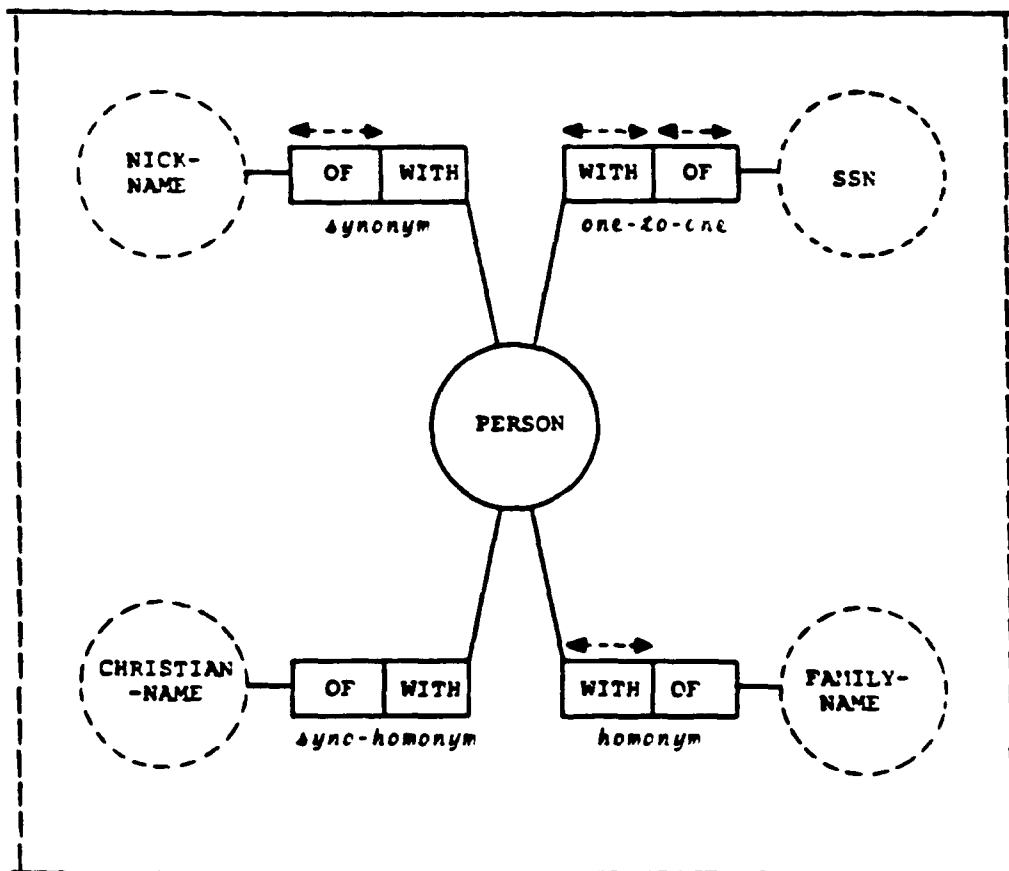
AN EXAMPLE

The next page contains a small binary semantic information model. Some of the sentences described by this model are:

"A user knows his problem area.

A user makes an information model which describes the problem area.

An information model is reviewed by an information analyst and may be implemented by a data base.

An information expert may be an information analyst or an information engineer or both.

Before a user can make an information model, he must know his problem area and attend a training course.

An information analyst must have experience."
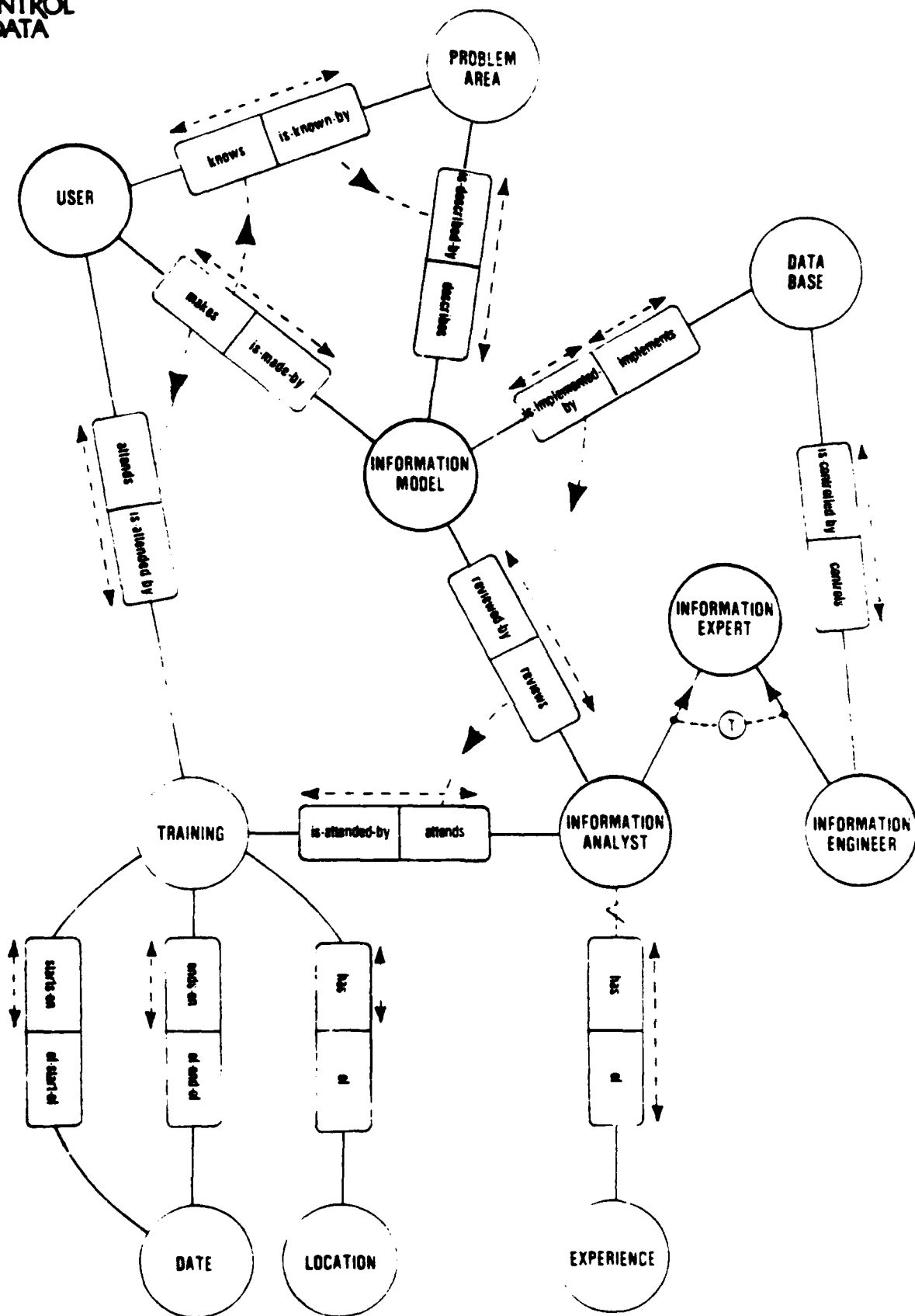
QUERY PATH/CONSTRAINT

All legal query relationships and accesses are defined
in the model. Invalid query paths cannot occur.

This provides two advantages:

1) a high level and very natural interface
   to the model information and

2) rejection of invalid accesses.

EXAMPLE: HIGH LEVEL NATURAL UNDERSTANDING

LIST EXPERIENCE of the INFORMATION ANALYSTS
       reviewing the INFORMATION MODEL
       describing the PROBLEM AREA
       "Manufacturing Automation."

```
SELECT EXPERIENCE FROM ANALYST-EXPERIENCE
WHERE PERSON # IN
 SELECT PERSON # FROM INFORMATION-ANALYST;
 WHERE PERSON # IN
  SELECT INFORMATION MODEL-ID FROM REVIEWING
  WHERE INFORMATION-MODEL-ID IN
  SELECT INFORMATION-MODEL-ID FROM INFORMATION
     ...
     WHERE PROBLEM-AREA-DESCR =
     "Manufacturing Automation"
```

```
INFORMATION ANALYST
+----------------------+
: Person #             :        P21       P22       P23
: -------------------- :
: Residence location   :        Mpls      Boston    Seattle
: Work location        :        Mpls      NY        Seattle
: Qualification        :        Cnslt     Analyst   Cnslt
+----------------------+
```

Valid — List all information analysts whose residence
location is the same as their work location.

Invalid — List all information analysts whose quali-
fication is the same as their work location.

Resolution of Ambiguity
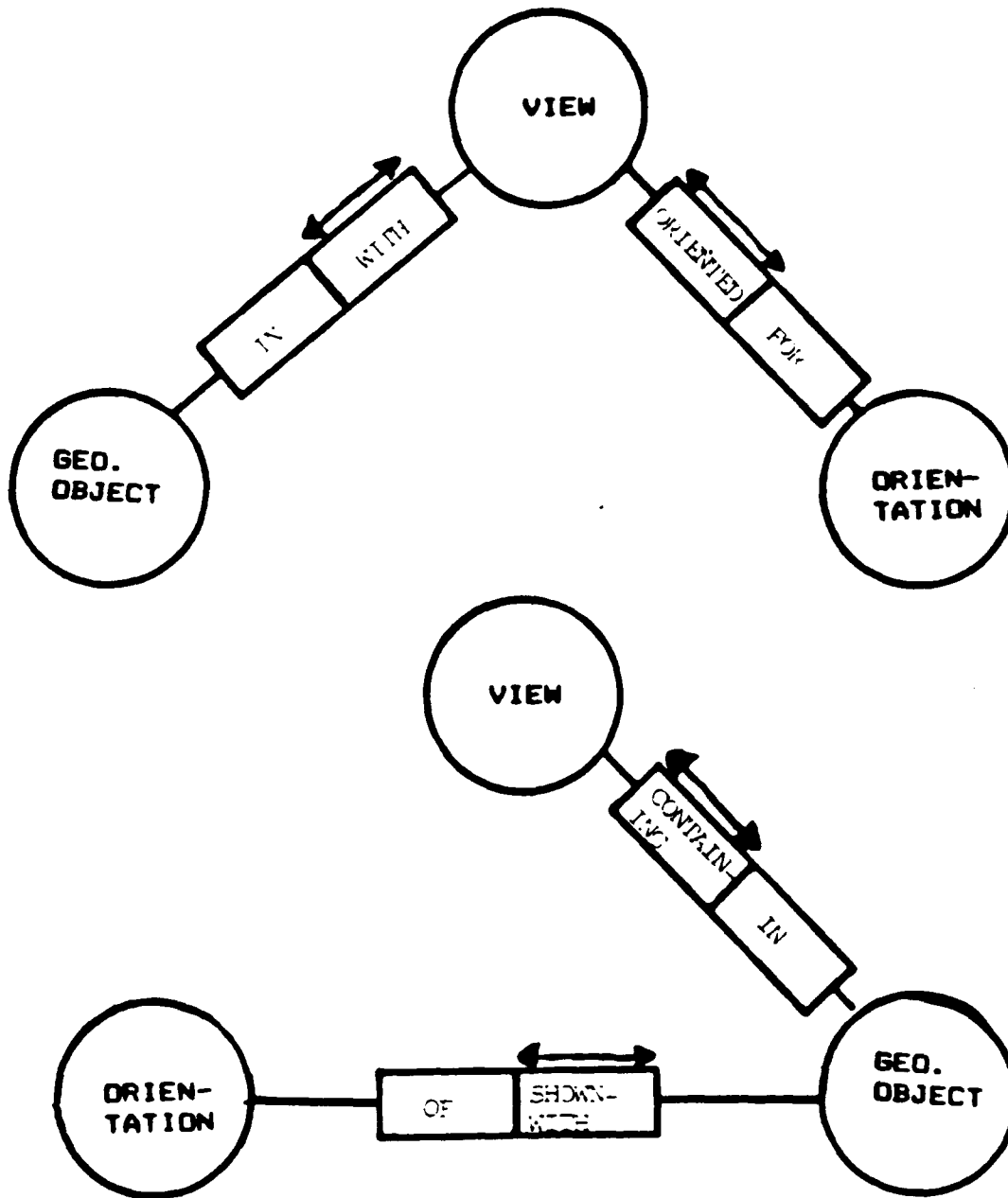
Constraint Expression

Integrity rules are captured naturally. Other techniques require artificial structures which in many cases still don't handle all the constraints.
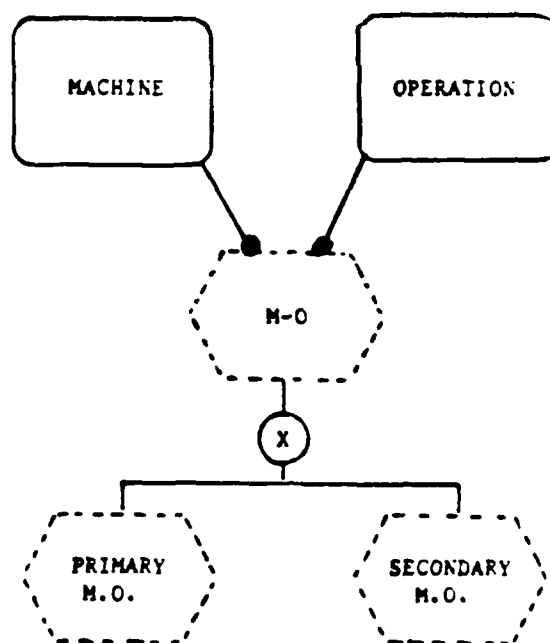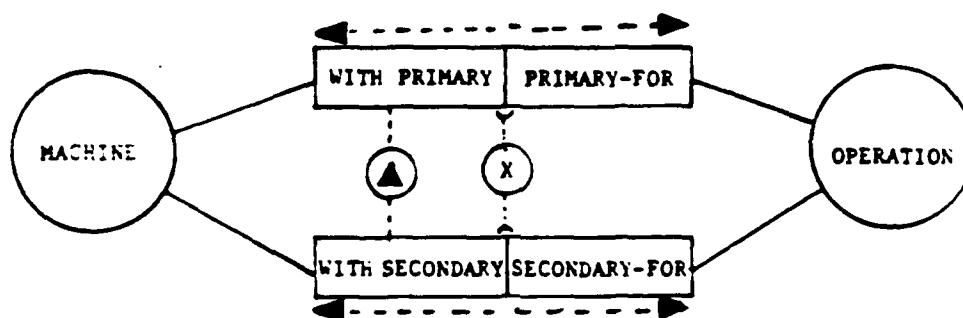
Scope

A scope is an abstraction tool.
The scope concept serves the user to abstract from details while studying an
information system grammar and to concentrate attention on the interesting
part of it.
A scope is a conglomerate of concept types, symbol types, idea types, bridge
types, constraints, and scopes, belonging together according to user
semantics.
A scope can be specified in terms of other scopes. There are no restrictions
to the number of concepts and the overlap of scopes in scopes.
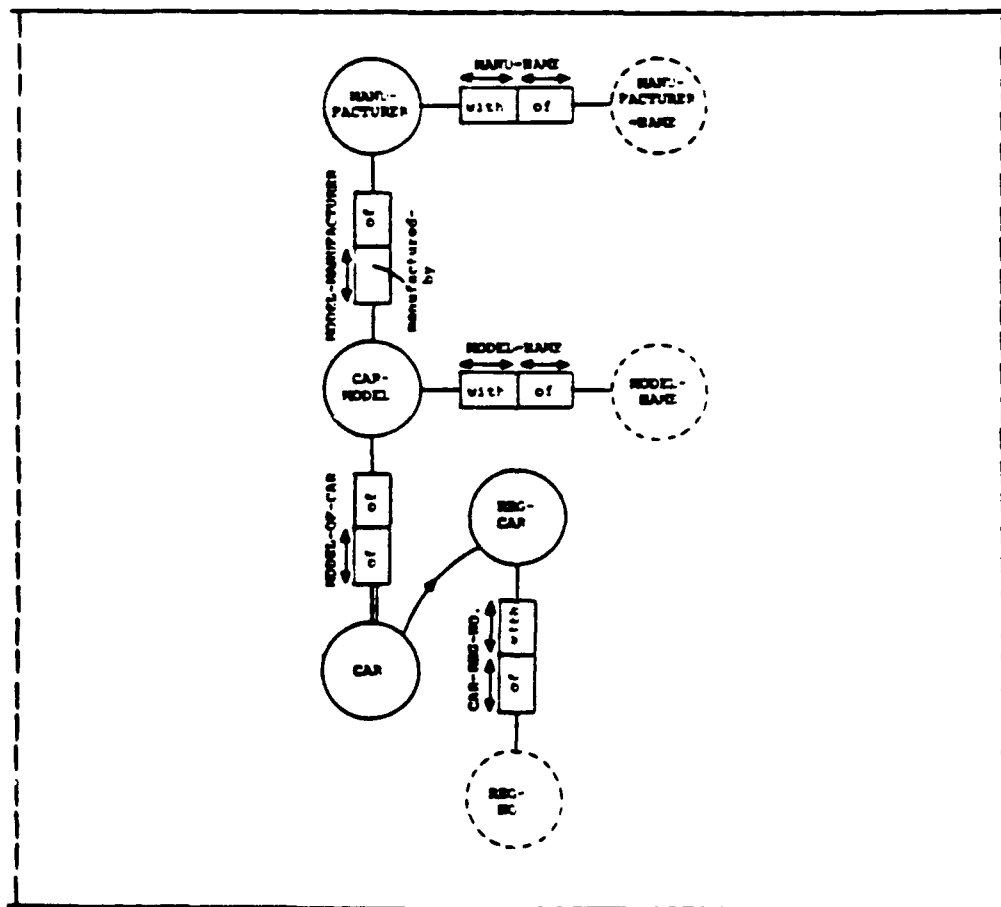
Zoom

A zoom is an ordered set of scopes, each lower level being more detailed than a higher level.
Therefore, any concept present at a certain level of a zoom must, at least, exist as such on all lower levels of that zoom.

Like a scope, a zoom is a powerful abstraction tool.
but the zoom mechanism adds another dimension: a scope can be seen as a way to stake out a (flat) area of special interest. Suppose that a certain scope coincides (probably not by accident) with an intermediate level of a certain zoom; then it is possible to zoom in and out on that scope.
In this way it is possible to observe a certain area of interest from a distance, without its (irrelevant) details, or, on the other hand, to observe it in great detail.
The advantage of a zoom over a collection of scopes is that the zoom mechanism guarantees consistency between the levels.

An idea/bridge schema illustrating the grouping options
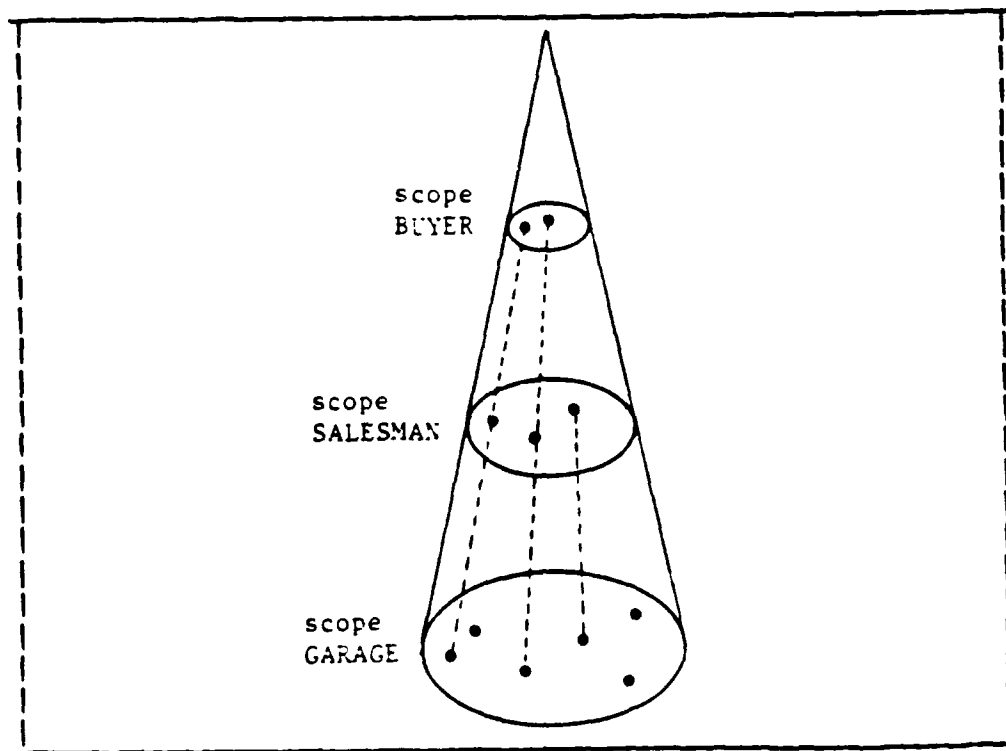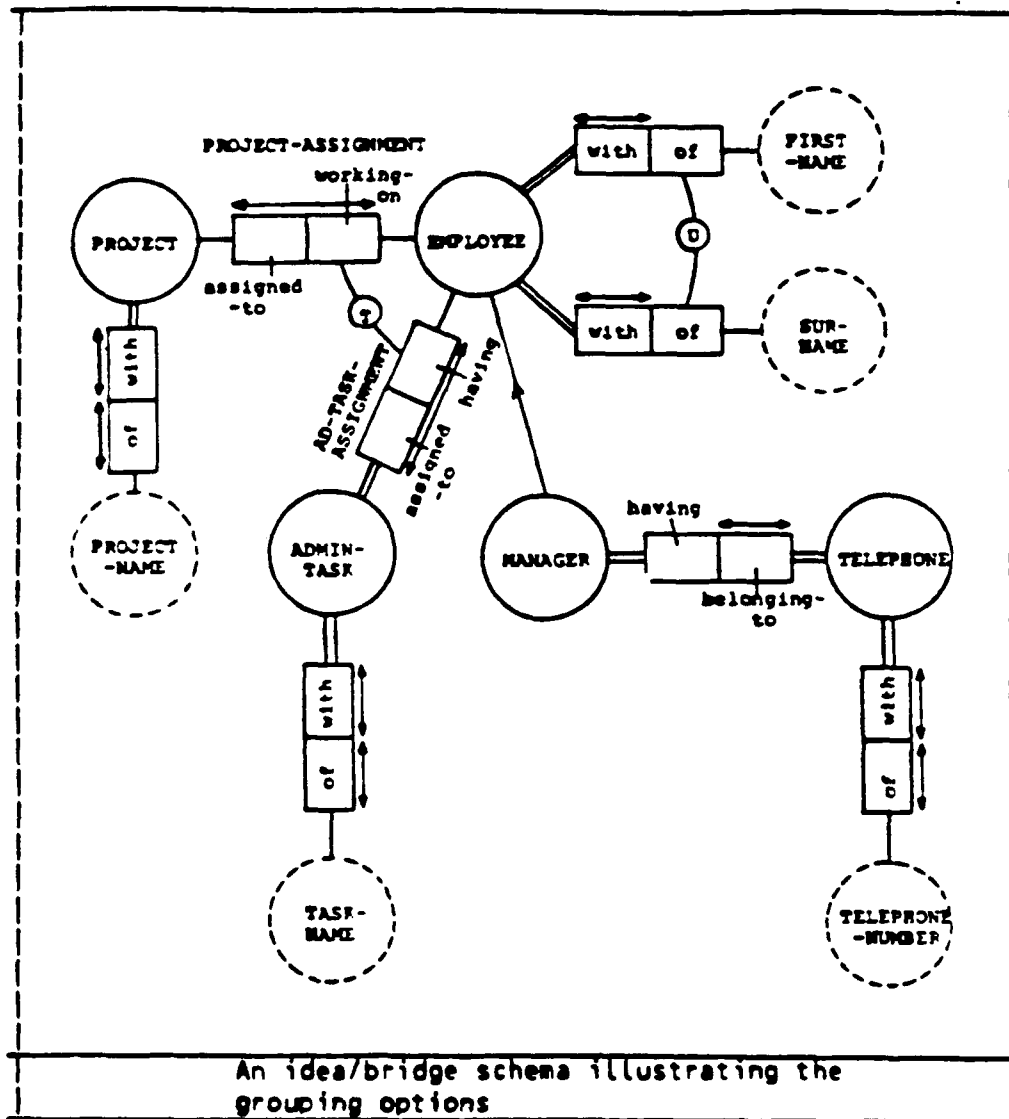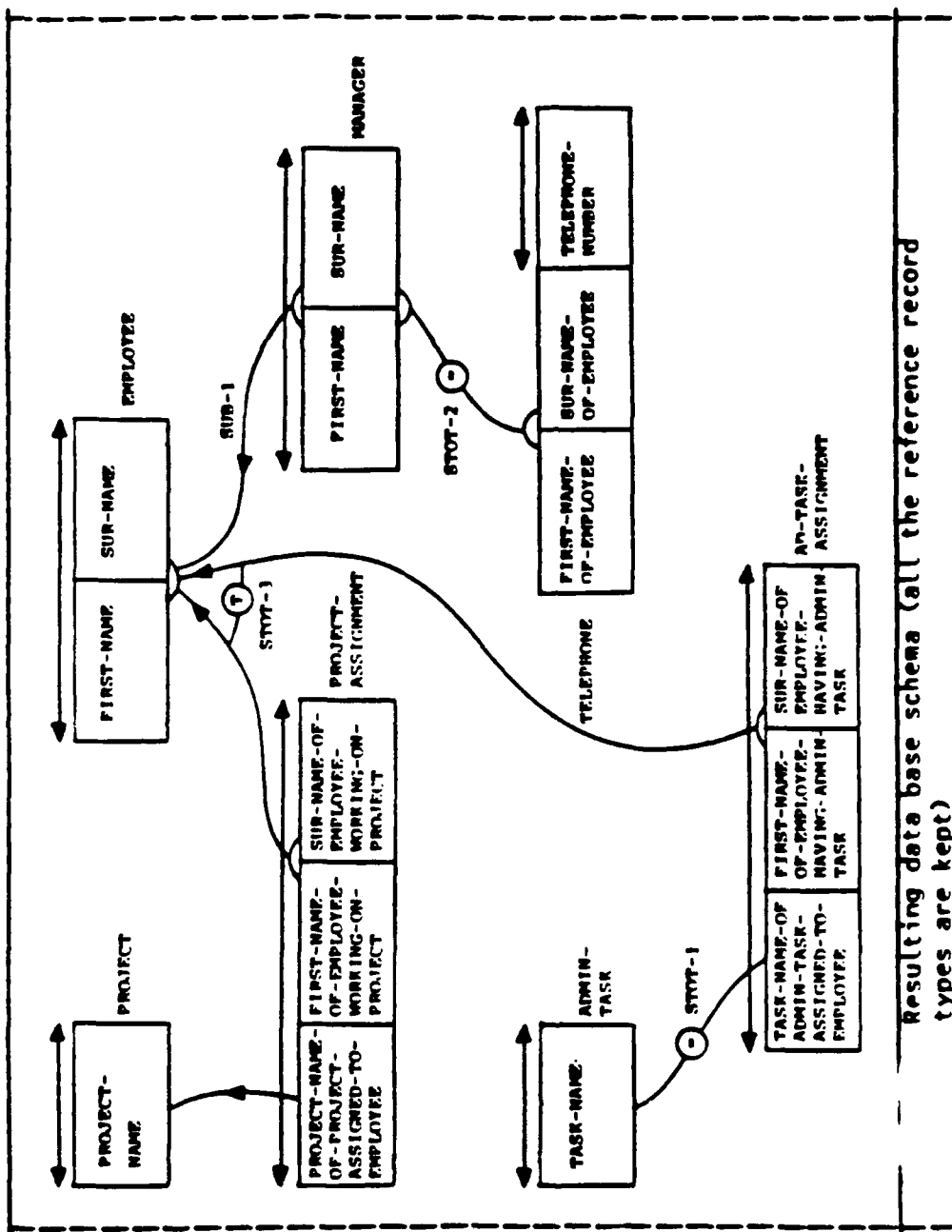
Resulting data base schema (all the reference record types are kept)

EXPRESSIVE POWER

ISO TC97/SC5/WG3

```
----------------------------------------
:                                        :
:   Interpreted Predicate Logic          :
:                                        :
:   -------------------------------      :
:                                 :      :
:   Binary Semantic               :      :
:   Network                       :      :
:                                 :      :
:   ----------------------        :      :
:                       :         :      :
:   Entity Attribute    :         :      :
:   Relationship        :         :      :
:                       :         :      :
----------------------------------------
```
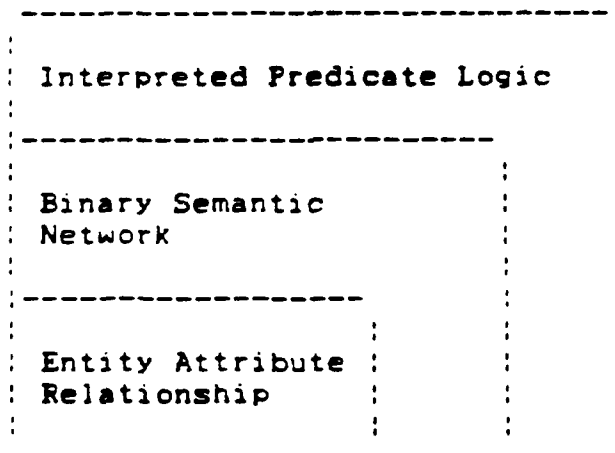
	EAR	BSN	IPL
Propositions modeled	25	40	47
Based on	Data	Semantics	Math
Graphical notation	X	X	
Tools	X	X	

FEATURES OF NIAM

o   Founded in logic and set theory with strong links to linguistic theory.

o   Provides a feature−rich semantic modeling technique for high quality modeling efforts.

o   Permits rigorous analysis and specification by user experts in a language close to their problem description.

o   Provides mathematical and linguistics checks on the correctness of the model and makes visible specification of (hidden) assumptions.

o   Maps easily to data specifications; this has been automated for normalized models.

o   Has international recognition at ISO and IFIP levels. Is a true ANSI/SPARC conceptual schema.

o   Proven in more than ten years of real world experience.

CED
CONTROL
DATA

SOME ADVANTAGES OF NIAM

o User statements may be captured naturally with minimal re-interpretation by a "data expert."

o The technique encourages the specification of integrity rules which may be overlooked by other techniques.

o Mapping to the data model (normalization) is automated and does not have to be performed by the analyst. This avoids distraction during a user session.

o All relationships are shown and all views equally valid yielding greatest conceptual uniformity and simplicity.

o More documented semantics and knowledge rules yield more precision and less ambiguity. The Object-Role style of modeling promotes conceptualization.

o Some university and industry experiments indicate a higher degree of consistency than with other techniques.

o The technique is stable, others are still evolving.

RECAP

The NIAM Binary Semantic Information Model consists of:

OBJECT TYPES – Fundamental categories of the real world;
divided into two kinds, CONCEPTS and SYMBOLS.

SUBTYPES – Flexible generalization links between the
OBJECT TYPES.

FACT TYPES – The basic business rules of the real world;
divided into two kinds, IDEAS and BRIDGES;
all relationships (connections) are shown.

INTEGRITY CONSTRAINTS – Integrity rules containing
knowledge of valid system states and transitions;
basic SET ORIENTED constraints and extended
LOGICAL PREDICATE constraints.

QUERY CONSTRAINTS – The definition of all legal queries
or processing access to the knowledge.

REFERENCES

1. "Informatiesystemen en De Volgende Generatie Data Base Management Systemen", G. M. Nijssen, in *Informatica*, Amsterdam, 1976.

2. "A Gross Architecture" for the Next Generation Data Base Management Systems", G. M. Nijssen, in the *Proceedings of IFIP-TC-2 Freudenstadt*, 1976, published by North Holland.

3. ISO TC97/SC5/WG3 - *Concepts and Terminology for the Conceptual Schema and Information Base*, edited by J. J. Van Griethuysen, 1982. (Available from ANSI.)

4. IFIP WG 8.1 - "NIAM : An Information Analysis Method", G.M.A. Verheijen and J. Van Bekkum in *Information Systems Design Methodologies: A Comparative Review*, North Holland, 1982.

5. "Modelling and Manipulating Production Data Bases in Terms of Semantic Nets", Robert Meersman and Frans Van Assche, Control Data, 1983.

6. "The RIDL Conceptual Language: An Abstract", Robert Meersman, Control Data, 1982.

7. ANSI/X3/SPARC - *Report of the Study Group on Data Base Management Systems*.

8. "The Automated Mapping from a Binary Concetual Schema to a 5th NF Data Base Schema", Frans van Assche, Danielle Simons, and Marleen Vanhoedenaghe, Control Data, 1983.

9. "Experience with Information Analysis in the US." - Paul Thompson, Control Data 1981.

10. "A User-Driven Methodology for the Design fo CAD/CAM and Engineering Databases", Roger Thyr, Proceedings of AUTOFACT 6 Conference, 1984.

11. "IA Literature List", ICIAS, Control Data. (A reference of approximately 130 articles on Information Analysis.)

NATURAL LANGUAGE ANALYSIS,
INFORMATION MODELING,
AND DATABASE ENGINEERING

Paul Thompson
Control Data Corporation
Minneapolis, Minnesota

Abstract

An overview of some concepts in knowledge engineering is presented:
How to analyze the content of user expert natural language; how to
build an Information Model semantic network; and how to transform
the Information Model into a database design.

1.   INTRODUCTION

The introduction of database technology in the 1960's caused a great
change in the way information systems were developed.

Previously information systems were set up to maintain their data in
process-oriented files. Some files simulated the manual systems.
Others were designed by programmers as scratch pads for data which
couldn't all be processed within the program during one computer
run.

Database technology, on the other hand, provided a single control-
ling mechanism containing _all_ of the data needed by an application.
And it stored the data in a way that it was easily accessible and
easily relatable. This new capability to get at data caused people
to think of data as having an existence independent from the pro-
grams which stored or processed it.

After initial experience with database technology, people realized
two facts:

  1. Database technology is not just a new and better file access
technique. It is a radically different approach to information
system development.

  2. To make progress it is necessary to rethink concepts of data and
how it represents information and to rework our approaches to system
design.

This paper presents some of the rethinking which as taken place.

2.   NATURAL LANGUAGE ANALYSIS

2.1 The Purpose of Information Systems

Let's step back from the complexities of system design, program
development and database management, and look at the fundamental
nature of information systems.

Quick. What do payroll systems, accounting systems, engineering
documentation systems, and clinical pathological laboratory systems
have in common ?

They are all information systems.

They all store and process information for the benefit of their
users.

They all store selected, pre-determined categories of information of

interest to the organization and select, combine and present subsets of this information at a time and place remote from the capture of the information.

Information consists of facts, facts about the "real world" and events that have happened in it, and conclusions that people have drawn about it.

Fundamental Fact 01: The purpose of any information system is to store facts.

An airlines information system stores facts about airplanes, passengers, flight schedules, fares, bookings...

A clinical pathological laboratory system stores facts about patients, doctors, machines, tests that have been ordered, samples that have been received, test results, QC results, and so on.

Based on these facts application programs can select, arrange and present useful information. For example, a list of patients ordered by the ward and room, preprinted labels for specimen collection, trend analysis of a patients' glucose level or the performance of a SMAC 20 (a laboratory machine which can make multiple tests on one blood sample).

2.2 The Use of Information Systems

There are always at least two users of any information system. For organizational information systems, this fact is obvious. For private information systems - an individual's notes, memories, files, etc - we can assume that he plays the roles of two different users first when he stores and then later when he retrieves the information.

Fundamental Fact 02: The users of an information system establish a dialogue with each other through the stored facts.

User 1 will input a fact and at a later time User 2 will make use of this fact. (See Figure 1.)

This principle is so fundamental and obvious that it is often overlooked, but the consequences are very important. If User 2 does not understand what User 1 is saying when he enters his facts then User 2 cannot make use of them or may misinterpret them and may take mistaken actions.

Even in the same discipline users frequently misunderstand each other; and it is a common experience for someone new to the discipline to go through a learning curve of a few weeks or months before he understands the "jargon" of the discipline. Think about the time required to become a doctor, and how difficult it is for a doctor to talk technically with a patient.

Think also about how many ordinary subjects like sports, politics, religion, and business in which we misunderstand what another says.

2.3 All Communication Regarded as Natural Language Sentences

There are many ways User 1 can communicate facts to User 2. He can speak to him, write a message, make a sign or symbol (like STOP), use body language, Morse code, write a mathematical formula, fill in a data collection form, type a message at a computer terminal, draw a computer flowchart, paint a picture, etc.

Because of the many varieties and complexities of communication, we select only one way to concentrate on as the basis for our work. This is communication in natural language sentences (written or verbal).

Fundamental Fact #3: All communication between a user and an information system can be regarded as a simplified form of natural language sentences.

(This includes textual, tabular and graphical communication. These can all be expressed in natural language by examining each part and asking the question, "What does this mean?")

This principle was first enunciated (in the database world) in the middle 1970's by Dr. G.M. Nijssen, head of the IFIP (International Federation of Information Processors) working group on databases. He had the novel idea to regard information systems not as massive flowcharts of program logic and system processes, but as a vastly simplified model of human communication.

2.4 Ambiguity and the Need for Recovering the Mental Model

When humans communicate in natural language, they hope they will be understood. Unfortunately, communication is rarely perfect. When a message is subject to multiple interpretations it is said to be ambiguous.

Let's look at some examples of ambiguity. First we'll look at Natural Language ambiguity, then at information system ambiguity.

(a)Natural Language Ambiguity.

Consider the statement:

    BITING DOGS CAN BE DANGEROUS

What does this mean ? If your context (mental model) is canine hydrophobia, then it means dogs which bite can be dangerous to you. On the other hand, if your context has the human as the subject then it might mean if you bite the dog, it could be dangerous for both of you.

In general, the sender of a message has a mental image of the situation which is only imperfectly referred to by the communication message. In order to recover the correct meaning, the receiver must share the same mental image. Otherwise, he makes wrong "assumptions". Good communication occurs when the receiver provides feedback or queries the sender to discover the context intended.

(b) Information System Ambiguity.

Now let's take an example from a record keeping system. Suppose we have the following data on a form. To keep the example simple we will limit it to five data items. (See Figure 2.)

Many people, particularly programmers and designers of information systems, take the "data" for granted. It's obvious that this table represents data about doctors, patients, and diseases.

It's also very easy to design a data file to hold this information. The data management system can then search for any particular doctor or patient. Sorts can be used to report by patient or doctor or disease or year or nationality.

All looks well until we ask the question "But what does it mean ?"

It could mean a doctor named Galen treated a patient named Lucius Commodus for a disease named Malaria. Or it could mean Galen knew Commodus and Commodus suffered from Malaria but Galen had nothing to do with treating the disease. Or it could mean Galen wrote about Commodus' Malaria. Or even that Galen treated Commodus but it was Galen who had Malaria. Or....

Similar questions could be asked about who was a Greek (Commodus or Galen) and what does the year 130 signify in relation to the other data elements.

I think you can see there are innumerable interpretations among the data elements in this simple table.

The consequences of this ambiguity are that a user of the information system can draw the wrong conclusions; and that redundant and inconsistent data may be stored, with all its attendant costs of clean-up.

Fundamental Fact #4: In order for unambiguous communication to take place, both the sender and receiver of the message must share the same mental model.

If you are developing an information system, you should stop at this point and ask yourself, "Have we documented the common mental model or do the users and programmers just assume they understand each other?" Because if you haven't, you're going to get into trouble.

2.5 Two Templates for the Mental Model

What we need is a way of insuring that both communicating parties share the same mental model. What we would like is to make this mental model visible. Then we can document this model and communicate it publicly to prevent ambiguity and errors.

First we'll look at two methods people have used to analyze sentences. Then we'll choose one of them for documenting information structure.

(a) Grammatical Structure Model

One way of analyzing communication is to use English grammar to break a sentence down into its elementary parts of speech. This method has an advantage that it has been used for centuries and is taught to everyone in school.

For example, every basic English sentence consists of a subject noun phrase (NP) and a verb phrase (VP). Symbolically, we represent the sentence by the formula:

    S -> NP + VP

The noun phrase identifies the subject of interest, and the verb phrase tells what the subject does, is or has done to it. (The arrow means "consists of" or "is produced when".)

Let's take an example.
LUCIUS COMMODUS WAS TREATED BY GALEN
FOR MALARIA.

```
-------------------------------------------
: NP = Lucius Commodus                     :
: VP = was treated by Galen for Malaria:
-------------------------------------------
```

In turn each part of the sentence can be decomposed into smaller parts according to linguistic rules used by every speaker of English.

```
S    -> NP + VP
        (noun phrase + verb phrase)

NP   -> DP + N
        (determiner phrase + noun)

VP   -> Aux + MVP
        (auxillary + main verb phrase)

MVP -> V + (NP) + (MAN)
        (verb + optional noun phrase
              +  optional manner phrase)
```

(b) The Object-Role Model

Another way of analyzing communication is to use the object - role
model. In this model, every sentence is regarded as the combination
of one or more objects which play parts (roles). This model gives
equal weight to all objects, and doesn't single out any one of them
as the "subject". Also, this model requires the class or type of
each individual object to be explicitly named.

The object-role model breaks a sentence down into its elements of
meaning.

Let's take the same sentence.
LUCIUS COMMODUS WAS TREATED BY GALEN
FOR MALARIA.

```
      object type 1   PATIENT
      object 1        Lucius Commodus
      role 1          suffering

      object type 2   DOCTOR
      object 2        Galen
      role 2          treating

      object type 3   DISEASE
      object 3        Malaria
      role 3          afflicting
```

This model has several advantages over the grammatic structure
model:

- it's independent of English language specific grammar hence it's
valid for all speakers

- it's independent of the order of the objects. ("Galen treated
Lucius Commodus for Malaria" is recognized as the same fact.)

- it communicates both the template from the mental model as well as
the particular instances

Language is a vehicle for communicating facts and creating understanding. It uses vocabulary and grammar and poetry as a complex envelope for carrying its message. When we need to analyze the envelope (the form of the message), we will use principles of grammar. But when the message itself is important, when we want to get at the main thoughts or ideas that people want to express and that lie at a deeper level, we will use the object role model.

Therefore, because we are interested in the content of the message, from now on we will use the object - role model as the basis for documenting the mental model. In the next section we will present a formal definition of the basic object-role model and extensions to it which have proven very useful in practical applications.

2.6 Population Table

The population table is the graphical means for illustrating the object role model. It clearly distinguishes between the general template and the particular instances. It gives the opportunity to all additional instances if necessary to confirm understanding.

(See Figure 3.)

In the population table, the object types are in circles connected to the roles. The object instances are in the body of the table. Population tables are smaller than data tables because they are only made for elementary sentences.

2.7 Elementary Sentences

A sentence is elementary when it cannot be decomposed into smaller sentences without loss of information or introduction of ambiguity. Each elementary sentence contains a single indivisible message, a single fact about the "real world".

A compound sentence conveys many facts. It is composed of two or more elementary sentences. The problem with compound sentences is we don't always know how they are put together. The receiver doesn't always know the rules of composition. Thus they are more susceptible to incorrect interpretation.

We can avoid these problems by always decomposing compound sentences into elementary sentences. In our example there are three different elementary sentence types. (See Figure 4.)

2.8 100% Prescription

Finally, we mention Fundamental Fact #5: 100% of the user's information problem can and must be predefined in a written, formal Information Model.

The information system can enforce correct usage and prevent

pollution and nonsense only if it knows all of the rules the information must obey.

We have seen that numerous interpretations are possible to people with different mental models. The administrator who specifies the system may have one set of meanings, the doctor who queries the information system may have another assumption, and the laboratory technician who enters the data may have a third interpretation.

And that means someday, someone could make a very serious mistake. In a medical situation, it could cost a life. A computer can do nothing to prevent the situation until it is provided with the correct Information Model.

All interpretations are equally correct until there is one commonly agreed upon, documented mental model.

3.   THE INFORMATION MODEL

3.1 What is an Information Model

An Information Model is a blueprint for understanding a user's information world. It is both an abstraction of the real world and a prescription for any computerized information system.

An Information Model is a formalized object-role model . It contains specification of all object types, fact types (and roles) and constraints organized in a semantic network. It documents the user's mental model.

3.2 Why Needed

The main purpose of an information model is to document an understanding of the real world in order to permit unambiguous communication to take place. When two communicating partners share the same mental information model, then they both interpret the communication in the same way.

A second important purpose of the Information Model is to organize locally discovered facts into a global network, showing how they are related to previously understood information.

3.3 Concepts of the Information Model

There are two kinds of object types shown in the Information Model: entities and symbols.

(Refer to Figure 5 to see examples of these symbols.)

Entity Type - a solid oval representing all possible instances of some real world objects or concepts.

Examples: Doctor, Patient, Treatment, Bed-Assignment, Specimen, etc.

Symbol Type - a dashed circle representing the set of symbols which can refer to, or identify entities.

Examples: Doctor-Name, Patient-Nr, Treatment-Code.

In general, there is not a one to one relationship between symbols and entities. You will find some entities with more than one symbol used to refer to them (for example Doctor-name and Doctor-number), and some entities with non-unique symbols (for example, there may be several unconscious "John Does" in the emergency room.), and some entities and concepts with no officially recognized identifying symbols (for example, a lab technician's visual inspection of a testing machine or the informal group of workers playing cards at lunch).

Subtype - a directed line-segment connecting two Entity types, pointing from the subtype to the supertype. All instances of the subtype are automatically instances of the supertype, and all properties of the supertype are inherited by the subtype.

For example, all Doctors are Health Care Personnel, and share their properties, such as concern for patients health, assignment to hospitals, state licensing, etc. In addition, doctors have properties not possessed by the supertype, for example, the authorization to prescribe treatments.

Frequently, subtype relationships occur in families. This reflects the human mind at work classifying sets of entities. In Figure 5 there are two families headed by PERSON and by TEST.

Role - the part played by an object type in a relationship. Each object in the relationship plays a role.

For example, if a laboratory technician performs a test on a specimen, then the two roles are performing and tested. The technician is the performing technician and the specimen is the tested specimen. If the technician later does a quality check on a test result, then that technician is the checking technician.

- There are two kinds of fact types: ideas and bridges.

Idea type - is an information bearing relationship between two Entity types. Each Entity type plays a role in the relationship. The roles are placed in boxes next to the corresponding Entity types. This relationship can be read in both directions.

For example, ,A Doctor (is) treating a Patient, or the Patient (is) treated-by a DOCTOR.

Bridge Type - is a naming relationship connecting an entity type with a symbol type. A bridge represents an arbitrary naming convention.

For example, PERSON with SSN.

Constraint - is a restriction on the allowable states of the information base and/or on transitions between them.

A constraint is a rule of behavior, either a business rule such as a patient may not occupy more than one bed, or a commonly accepted natural law such as babies may be born only to female patients.

3.4 How Constructed

An Information Model is constructed by connecting together the elementary pieces of information discovered during analysis. These elementary pieces do not have to be derived in any given way. They can be used as they are found, when they are found, from any source. This flexibility exists only when the pieces are truly elementary.

The Information Model is constructed graphically, usually on large sheets of paper. First objects and their subtype connections are drawn. Then ideas and bridges are connected to the objects. Finally constraints are drawn connecting objects and roles.

As the diagram is constructed its correctness and meaning are constantly checked by verbalizing the symbols in English and by using population tables.

There are two methods of gathering the information for making the diagram: the guru method and the archeology method. The guru method involves interviewing subject matter experts whose knowlege is so thorough that recourse to other sources is not necessary. It is the easiest and most common technique. The archeology method involves examining reports, computer listings, policies and procedures and other documents to extract the elementary facts and build a coherent information model from them. It is much more difficult and slow.

3.5 Interviewing

Interviewing is the most common technique used in the initial stages of building an Information Model. Users tend to be at ease explaining in English the information they need, the objects of their environment and the rules of information behavior. Later in the analysis they begin to understand the symbolic language of the Information Model diagram and can participate in their construction.

During the initial stages a professional Information Analyst is very helpful in guiding user discusssions, keeping them focused on the subject and explaining the techniques.

The most important questions he asks during the interviews are

"Tell me about ..."

"What does this mean ?"

These questions elicit English replies which are rich in semantic relationships (facts) about objects. The information analyst then breaks down sentences into elementary facts and diagrams them.

3.6 Example of Interview Notes

"I need to keep information about patients and doctors and laboratory tests.

"When a patient is admitted to the hospital we collect basic demographic information such as his name and social security number and sex and year of birth. We also record his date of admission and later his date of discharge.

"Patients are treated by one doctor. However, they may also be seen by other doctors, who may give advice. Only the patient's primary doctor may prescribe laboratory tests. And a patient must be assigned a primary doctor before he can be seen by other doctors.

"A doctor may order tests for a patient. There are various kinds of test he may order. Each test is performed by a lab technician. A doctor could order several different kinds of tests for a patient on the same day and may order the same test to be performed more than once.

"The lab technician carries out the tests. Lab technicians and doctors are both health care personnel, but no one can be both a doctor and a lab technician. The lab technician performs patient tests and quality control tests on his equipment. These are the only kinds of tests allowed. Some tests are run more than once and the results are judged by the technician who writes comments on the reports. Different test results for the same requested test are distinguished by sequence numbers.

"Each test can be performed by only one lab technician, be ordered by only one doctor and be for only one patient. While it is possible for doctors or lab technicians to be admitted as patients, our hospital does not permit them to order or perform tests for themselves.

"Furthermore, if a doctor is admitted as a patient he may not treat himself."

3.7 The Information Model Diagram

The results of this interview are documented in an Information Model diagram. (See Figure 5.) Please note that this model is only

for illustration. It is too simple to be realistic. In a real
project more analysis would take place and many improvements made.

An information model diagram is a pictorial representation of the
problem. It is processed by the right half of the analyst's brain.
This activity complements the interviewing or verbal stage of
information analysis which draws mostly on the left half of the
brain. The left half analyses the information and understands the
elementary pieces of meaning; the right half syntheses the facts and
understands the organization of knowledge.

3.8 Meta Rules

Meta rules are guidelines for the correct and consistent
construction of an Information Model. They represent accumulated
knowledge about knowledge engineering, and they help the designer of
an information model detect fuzzy thinking and construction errors.

Some examples of meta rules are the following:

- no isolated object types.

- every subtype family has one common ancestor.

- every role must be connected to one and only one object type.

- in any constraint or query which involves a comparison, the roles
compared must connect to the same object type or a subtype and
supertype in the same family.

- every fact type has a simple uniqueness constraint; if not, the
fact is not elementary or an object type has been omitted.

- in any constraint or query the path specification must be
unambiguous.

3.9 Distinction Between Role and Object

Many beginners initially have trouble distinguishing between a role
and an object. It's common to see a model with two objects DATE of
admission and DATE of TEST. In reality there is only one object
(DATE) and it plays two roles: (DATE) of admission of (PATIENT) and
(DATE) of (TEST).

Use of distinct object types for distinct concepts and common object
types for common concepts eliminates redundancies and prevents
nonsense.

For example, the query, "Give me a list of all PATIENTs whose DATE
of ADMISSION equals their DATES of discharge" is a sensible query.
It involves the comparison of the common object type DATE.

On the other hand, the query, "Give me a list of all PERSONs who
were born-in the same YEAR as their SEX is clearly nonsensical. It

involves the illegal comparison of distinct object types.

3.10 All Access Paths Shown

The Information Model has the interesting and useful characteristic that all logical access paths to any piece of information are shown. Unlike other methods, access paths are never eliminated for "efficiency" reasons.

Any retrieval query involves a path or combination or paths through the Information Model.

Let's look at three cases:

- legal queries - the query path(s) is unambiguous; this query makes sense.

```
--------------------------------------------------
: "Give me a list of all PATIENTs treated-:
: by DOCTOR John Kildare".               :
--------------------------------------------------
```

```
--------------------------------------------------
:"Give me the VALUES of all TEST-RESULTS :
:for all TESTs for PATIENT with NAME      :
:Mary C. Stewart".                        :
--------------------------------------------------
```

- ambiguous queries - the query path intended is unclear

```
--------------------------------------------------
:"Give me a list of all PATIENTs and their:
:DOCTORs". This could mean:               :
--------------------------------------------------
```

    a) PATIENT treated-by DOCTOR
    b) PATIENT also-seen-by DOCTOR
    c) or both
    d) or even some longer path involving
       intermediate objects

- impossible queries - there is no path
```
--------------------------------------------------
:"Give me a list of all PATIENTS who      :
:haven't paid their bills".               :
--------------------------------------------------
```

To answer this question you must extend the Information Model.

Queries are much easier to formulate, discuss, revise, understand,

and agree-upon when using the Information Model than with traditional program documentation.

3.11 Why Constraints

Constraints are formalized knowledge rules about natural laws or organizational policy. They are the rules of behavior which are enforced when information is added to or deleted from the information base. They are also the conditions which hold among all information facts in the base. They guarantee the quality , usefulness, and consistency of the information. With constraints no user can accidentally or deliberately introduce nonsensical or inconsistent information.

Traditionally, checks for the correctness of the information base have been buried deep inside programs, unreadable and unverifiable. This situation has led to rather haphazard constraint enforcement. The amount of constraint checking has tended to be a function of the experience of the programmer, the ease of programming the constraint, and amount of time given to the project.

Users typically assume more constraint checking in their data than there really is. When converting a typical production system to one with enforced constraints they are amazed and distressed at the number of errors discovered in their production data.

3.12 Some Common Constraints

Information Analysts are unique in recognizing the importance of constraints and in developing methods and notations for assisting users in finding and specifying constraints.

It turns out that in a typical Information Model, construction effort, some 80% of the constraints uncovered fit into a small number of already recognized categories.

A few of the commonly occurring constraint types and examples are:

- subtype exclusion - two or more subtypes are mutually exclusive. (A TEST may be either a PATIENT-TEST or a QC-TEST, but never both.)

- subtype total - the union of two or more subtypes equals the total supertype. (A PATIENT-TEST and a QC-TEST are the only kinds of TEST)

- role uniqueness - every fact in the Information Model has one or more roles unique. (A PATIENT is treated-by a unique DOCTOR. A TEST is uniquely for one PATIENT.)

- role subset - the population of role1 must be a subset of the population of role2. This is a set algebraic formulation of constraints involving the concepts of "before" and "implies". (Before a PATIENT can be seen by other DOCTORs, he must first be treated by his own DOCTOR.)

- joint uniqueness - the combination of two or more roles uniquely determines an object. If you know the PATIENT, and the DATE, TIME, and KIND of TEST, then you know which PATIENT-TEST. (On the other hand QC-TEST must be determined in a different way because no patient is involved.)

- equivalence of path - two different paths from object type 1 to object type 2 give the same result. (The DOCTOR treating the PATIENT must be the same DOCTOR ordering the TESTs for the PATIENT.)

3.13 ISO

Recently the International Standards Organization conducted a study on the "Concepts and Terminology for the Conceptual Schema." ISO's Binary approach is essentially the Information Model described here.

4.   DATABASE ENGINEERING

After the information problem has been analyzed, the next step is to design the database. A good database design will be easy to understand, modify and enhance, high in performance, secure and reliable, easy to use, and meet the needs of its users.

4.1 The Data Model

The designer of a database expresses his design in terms of a data model. A data model is a map of the data organized into a network of record types. Each record type contains one or more named data elements. A data element is the smallest addressable unit within a database. Solid lines link together data elements in different record types, or connect from "out of the blue" into a data element. These paths into and through a data model allow the user to find the data he needs. Advanced data models also contain data integrity constraints. (See Figure 6.)

4.2 Why Needed

The database user has several reasons for needing a data model as he plans his use of the data.

His most fundamental need is to know the names of the data elements which are documented in the data model so he can tell the computer which data to access.

(Because so many data elements have the same format, it is not possible to distinguish them just on the basis of their representation. Consider for example the date 01/05/84. Is this the Date of admission, the DATE of discharge, or the DATE of birth ?)

Secondly, the database user needs to know which data elements are organized together into the same record type. Because most computer

languages access data one record type at a time, it's important to know which data elements will be available together.

Thirdly, the database user needs to know the semantic paths through the data model in order to retrieve data related in a given way to the data he already has.

(These are not physical paths such as the CODASYL network structure, but are the basic understanding of meanings needed to ask intelligent questions.)

It is in this last aspect of the need to know the path or paths to related data that the user is likened to a navigator and the data model to a map. In planning complex data applications, the data model map is an indispensible tool for the data navigator.

4.3 The Schema

The computer's database management system (dbms) also needs to know the data model. A description of the data model is written in a computer syntax called a database schema. The dbms reads, checks and safely stores away the database schema. Then it uses the schema to plan how to organize and retrieve the data efficiently, and to guarantee its correctness.

Whenever the user asks for an update or retrieval, the dbms consults the schema first before accessing the data. In this way it can carry out the users' commands correctly and efficiently.

4.4 Efficient Implementation

Efficient implementation of large databases on present day computing machines requires a grouped data model. The data model is formed by grouping together the concepts of an information Model into efficient record types, data elements and their connections.

(An ungrouped Information Model, though more semantically rich, more detailed, and more flexible, requires advanced computer architecture for efficient implementation.)

Two cost characteristics of today's computers are important considerations for the grouping algorithm:

- Storing large amounts of data is expensive. Compressing data to reduce redundancy decreases storage costs by 50% or more.

- Accessing a block of data from a storage device and bringing it into main memory is a slow and expensive process. However, reading the data elements from the block once it is in main memory is fast and cheap. Hence data elements which are used together should be grouped together into the same data block to reduce access costs.

4.5 The Grouping Algorithm

In the previous section we discussed that too few data elements grouped together result in an inefficient design. On the other hand, it is also well known that too many data elements grouped together result in redundant storage and complex programs. What we need is an intermediate amount of grouping which is just right.

The full grouping algorithm also includes steps for checking the Information Model for consistency and referenceability, for selecting some tuning options, for grouping the constraints, and for constructing the full data model. This algorithm is taught in Information Analysis seminars and has been computerized and implemented for about a half a dozen current database managers.

Here's the simple selection criterion used in the grouping algorithm:

For each entity class: select for grouping only directly connected, single valued facts.

This simple grouping rule generates record types which satisfy efficiency and ease of use criteria.

- A fact is stored only once (non-redundancy)

- A fact is stored in a fixed location (predictability)

- Every data element is determined by "the key, the whole key and nothing but the key" (normalization)

- No wasted storage space (low cost)

In short, an elegant method

4.6 Relationship to Logic Programming

Logic programming using languages such as PROLOG and LISP is a rapidly advancing new style of problem solving. Instead of specifying the flow of process as in a conventional programming language, the programmer specifies the relationships or predicates which hold and rules for deriving new knowledge.

Just like in the database world, it is necessary to first analyze the communication and build an information model of the information problem. Before logic and deduction can be applied we must first be sure that we understand which problem we are solving.

The predicates which are declared to the program are the elementary sentences or groups of elementary sentences from the information model. When grouping the sentences the same rule as before is used with the addition that the total role constraint is observed: only single valued facts which are connected to the object with a total

role constraint are grouped together. In this way so-called missing values are avoided.
.cp 5

4.7 The 3 Schema Architecture

So far we've talked as if each database system has only one schema. And while in fact many do, modern database management is evolving to a three schema architecture. This need was first recognized by ANSI.

4.8 Why Three Schemas

Database systems with one schema suffer from lack of data independence. Each user has the same view of the data. If any change is made to the data model, then all users will have to change.

In a two schema environment users have more data independence and protection. The database schema can change somewhat, and the users' subschemas stay the same. They continue to use and view data in their preferred subschema model.

In a three schema environment the database schema is replaced by a conceptual schema which defines the WHAT of the data problem, one or more internal schemas which define HOW the data is most efficiently stored and retrieved, and one or more external schemas which define HOW the user groups prefer to view and access their data. The conceptual schema is created automatically by the grouping algorithm. (See Figure 7.)

A database system engineered with a 3 schema architecture provides a number of direct and indirect benefits to its users. The direct benefits are

- Any user can alter his preferred view of the data described in the external schema without affecting any other user.

- A database engineer can completely reorganize the physical databases described in the internal schemas without affecting a single line of user source code or interactive procedures. In particular, the database engineer can tune the physical database structures following implementation.

- An enterprise administrator can turn constraints on or off and can enlarge his data model described in the conceptual schema to support new functions with no effect on existing programs, interactive procedures, or data organization.

Secondary benefits include the following:
- Less fear of change, because system-wide correctness is guaranteed by the conceptual schema.

- Smaller update programs, and much less development cost.

- Elimination of garbage from the data with constraint enforcement

- Avoidance of lock-in to current physical dbms technology.

Figure 8 shows the steps from problem recognition to 3 schema implementation.

5. CONCLUSION

5.1 Real World Experience

Since 1976 a group of Information Consultants have been applying the concepts of natural language information analysis, information model construction and data modeling to large, real world information system projects. In these nine years they have helped more than 120 clients develop modern effective information systems. In several cases, the clients had spent 5 years and as many million dollars in traditional analysis methods without results.

During this work, the natural language basis of factual information communication and the object - role model have been tested successfully in about 20 languages, including Indo-European, Semitic, and Oriental languages. This practical experience has given us confidence in the fundamental validity of our linguistic analysis.

5.2 Database Applications

Large scale application of the Information Analysis methodology has been to computerized database application systems for manufacturing, service and governmental organizations. Examples from various industries include a clinical pathological laboratory system, an engineering documentation and configuration management system, a combined turbine engineering design and drawing data base, a health physics records keeping system for a nuclear utility, a petroleum exploration database, and the entire application portfolio for two insurance companies.

5.3 Project Characteristics

The most striking characteristics of these projects are the increased user understanding of his requirements; his increased participation in and commitment to the requirements specification effort; and his satisfaction with the final implementation.

Typically, a user can become proficient with a few days' classroom training and learning experience in a project team environment. Thereafter, he takes a direct hand in the analysis and documentation work.

In these efforts, Information Analysis has found more requirements, and specified them more precisely than traditional analysis methods,...sometimes dramatically more. As a result the implemented

applications are more complete, have fewer errors, and require less
programming rework both during initial construction and after
implementation. Database design, previously extremely difficult, has
become more routine.

Data processing professionals report greater productivity, users
receive more flexible and complete solutions, and management sees
better cost and schedule control.

5.4 Wider Applicability

But Information Analysis is not limited just to application
development. It has been applied to the analysis of a user guide to
find unclear passages, to the analysis of industry standards for the
interchange of graphical computerized drawings, to a work
environment prior to the introduction of office automation, to the
design of a database manager and an information dictionary, and to
the analysis and improvement of development methodologies including
itself.

5.5 The User Expert as Developer

When computers were first introduced, the first programmers were
user experts who had to learn the language of the computer. Later,
when computers became more complex, specialists were trained in the
intricacies of computers. These specialists knew a lot about the
machine but very little about the business and information needed to
support it in user areas. Worse, they built computer software upon a
mental model of a machine (disks, tapes, records, keys, formats,
pointers, pages, sectors, etc.) foreign to the user and hence a
barrier to development.

Today with Information Analysis we are seeing a change in how
systems are developed. The user expert is now taking charge of
analysing his own problem and giving his specifications as require-
ments to the data processing shop. He is no longer accepting a
passive role as the victim of automation; today he is controlling
it. As a consequence, the nature of the software development process
is changing. In this process the professional information analyst
acts as an educator, reviewer, guide, and friend.

REFERENCES

1. "Informatiesystemen en De Volgende Generatie Data Base Management Systemen", G. M. Nijssen, in Informatica, Amsterdam, 1976.

2. "A Gross Architecture for the Next Generation Data Base Management Systems", G. M. Nijssen, in the Proceedings of IFIP-TC-2 Freudenstadt, 1976, published by North Holland.

3. ISO TC97/SC5/WG3 - Concepts and Terminology for the Conceptual Schema and Information Base, edited by J. J. Van Griethuysen, 1982. (Available from ANSI.)

4. IFIP WG 8.1 - "NIAM : An Information Analysis Method", G.M.A. Verheijen and J. Van Bekkum in Information Systems Design Methodologies: A Comparative Review, North Holland, 1982.

5. "Modelling and Manipulating Production Data Bases in Terms of Semantic Nets", Robert Meersman and Frans Van Assche, Control Data, 1983.

6. "The RIDL Conceptual Language: An Abstract", Robert Meersman, Control Data, 1982.

7. ANSI/X3/SPARC - Report of the Study Group on Data Base Management Systems.

8. "The Automated Mapping from a Binary Conceptual Schema to a 5th NF Data Base Schema", Frans van Assche, Danielle Simons, and Marleen Vanhoedenaghe, Control Data, 1983.

9. "Experience with Information Analysis in the US." - Paul Thompson, Control Data 1981.

10. "A User-Driven Methodology for the Design fo CAD/CAM and Engineering Databases", Roger Thyr, Proceedings of AUTOFACT 6 Conference, 1984.

11. "IA Literature List", ICIAS, Control Data. (A reference of approximately 130 articles on Information Analysis.)

BIOGRAPHY

Mr. Thompson is a Principal Consultant with Control Data responsible for seminar development, client consulting, and methodology development and introduction. He has supported Information Analysis projects for the last eight years. Prior to that he has had 14 years experience in scientific and commercial data processing. Mr. Thompson has a B.S. from Yale University and an M.S. in Computer Science from Purdue University.

(An earlier version of this paper was presented at the eighteenth annual International Conference on System Sciences, January 2-4, 1965.)
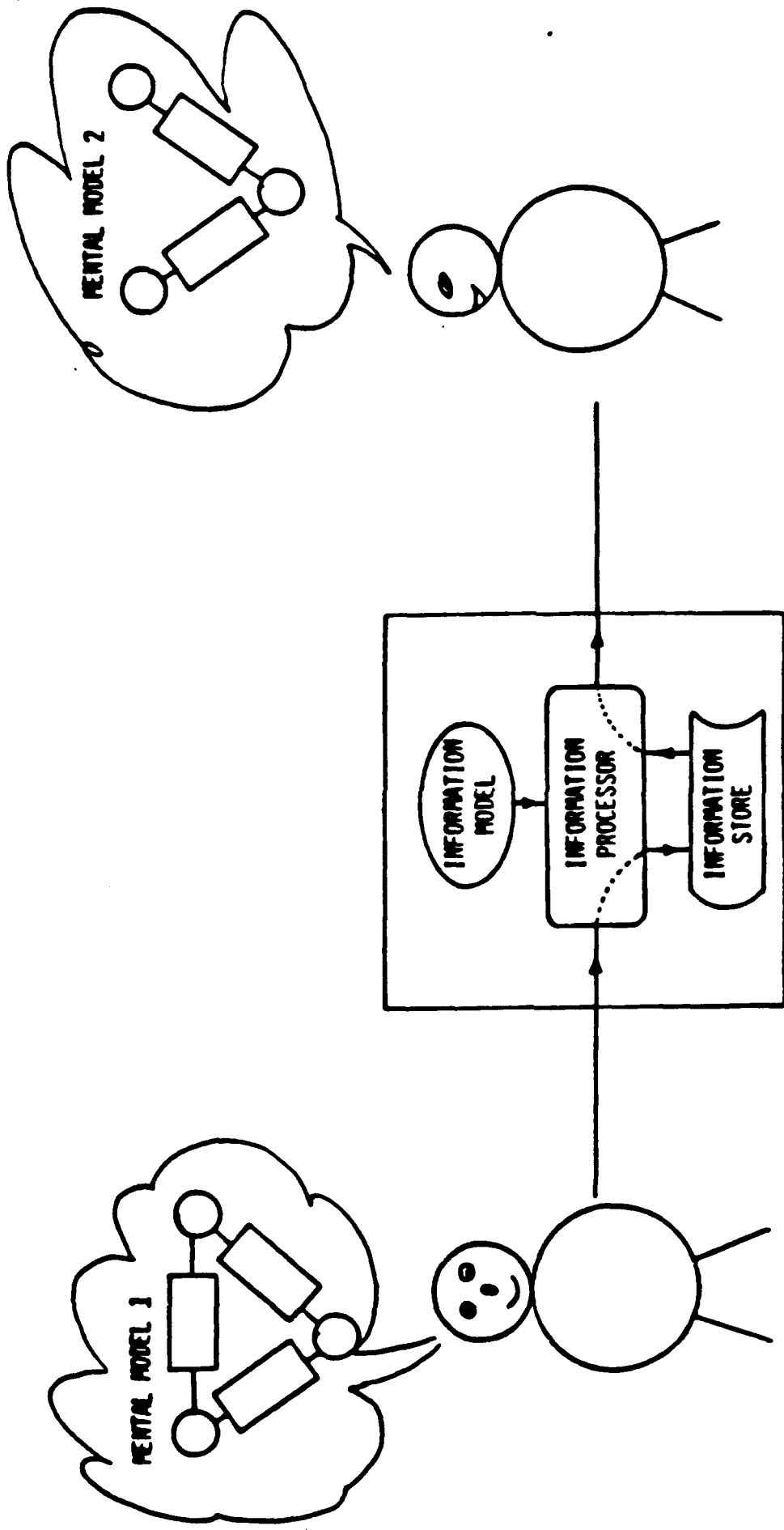
FIGURE 1. COMMUNICATION THROUGH AN INFORMATION SYSTEM

DOCTOR	PATIENT	DISEASE	YEAR	NATIONALITY
Galen	Lucius Commodus	Malaria	130	Greek
Galen	Marcus Aurelius	Sprain	130	Greek
Vesalius	Charles V	Chest cold	1514	Belgian
Harvey	Francis Bacon	Baldness	1578	English
Harvey	James I	Indigestion	1578	English

Figure 2.   Data Table
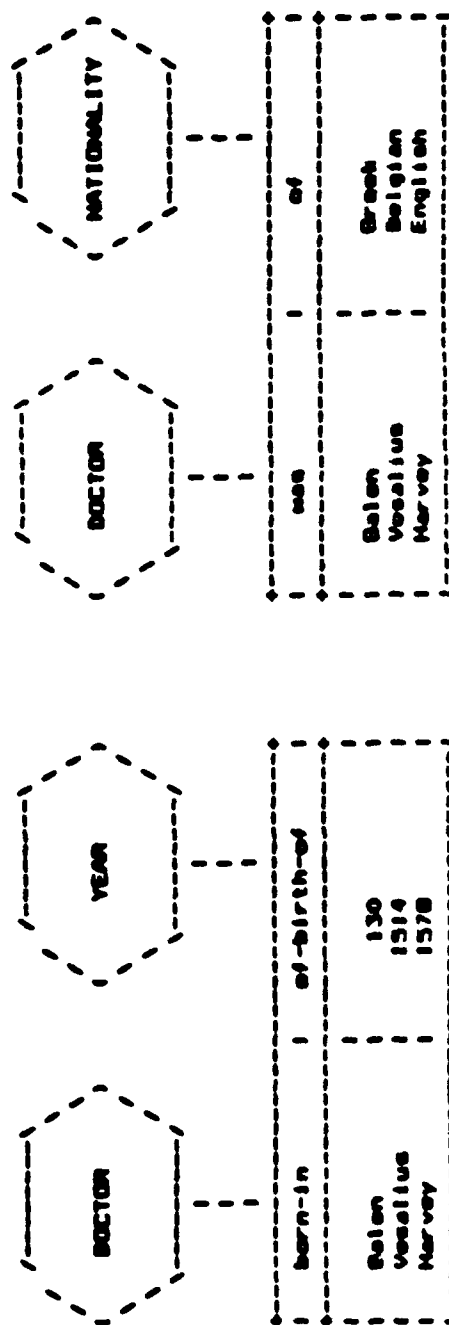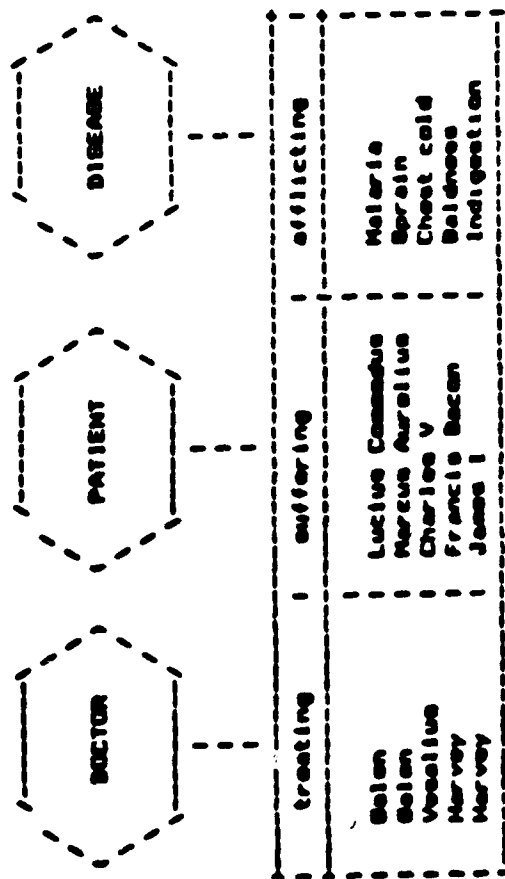
Figure 3. Population Tables

1. Galen   treated   Lucius Commodus for Malaria.
    Galen   "   Marcus Aurelius   " Sprain.
    Vesalius   "   Charles V   " Chest cold.
    Harvey   "   Francis Bacon   " Baldness.
    Harvey   "   James I   " Indigestion.

2. Galen   was   Greek.
    Vesalius   "   Belgian.
    Harvey   "   English.

3. Galen   was born in   130.
    Vesalius   "   1514.
    Harvey   "   1578.

        Figure 4.   Elementary Sentences
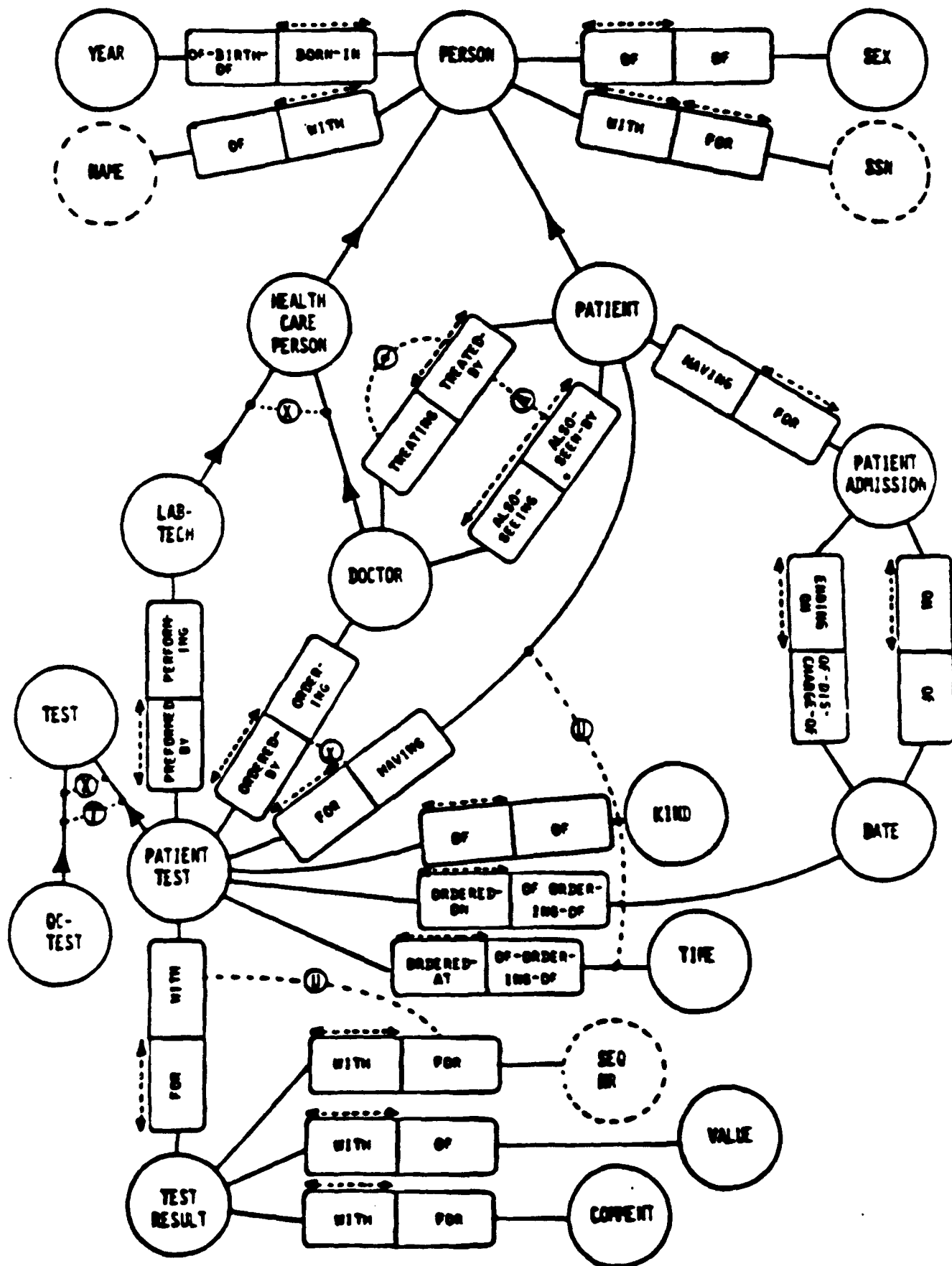
FIGURE 6. AN INFORMATION MODEL
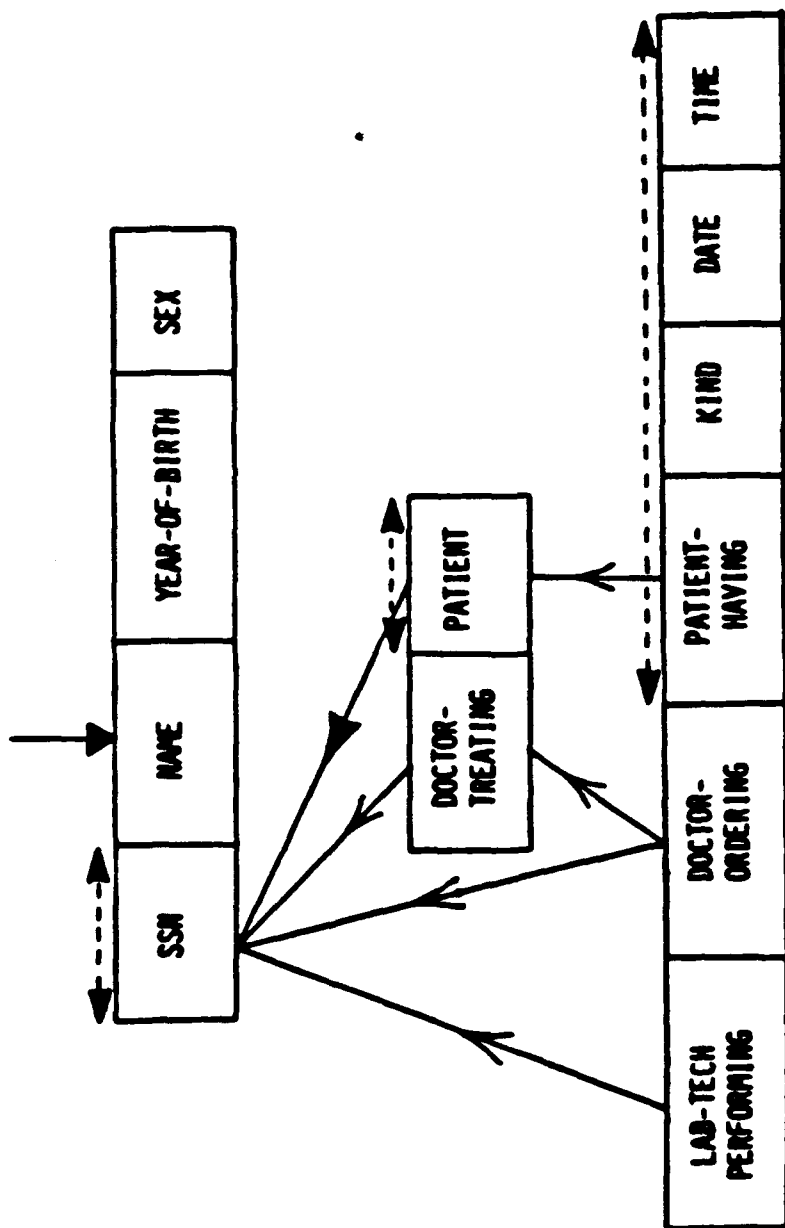
FIGURE 6. A PATIENT DATA MODEL
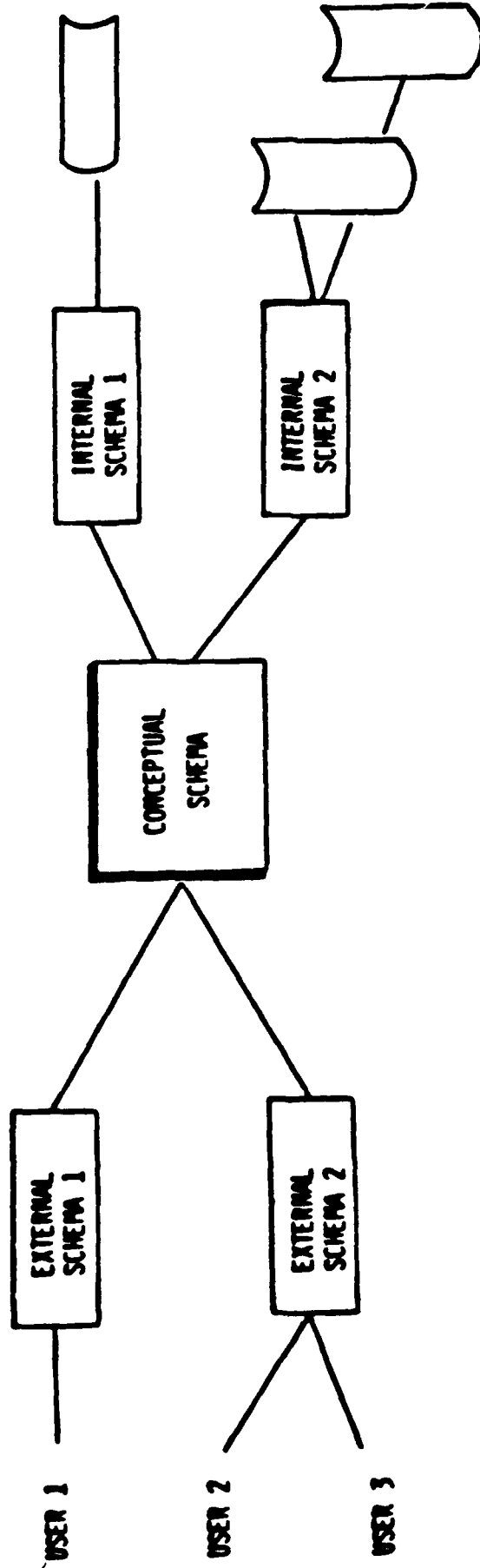
FIGURE 2. ANSI 3 SCHEMA ARCHITECTURE
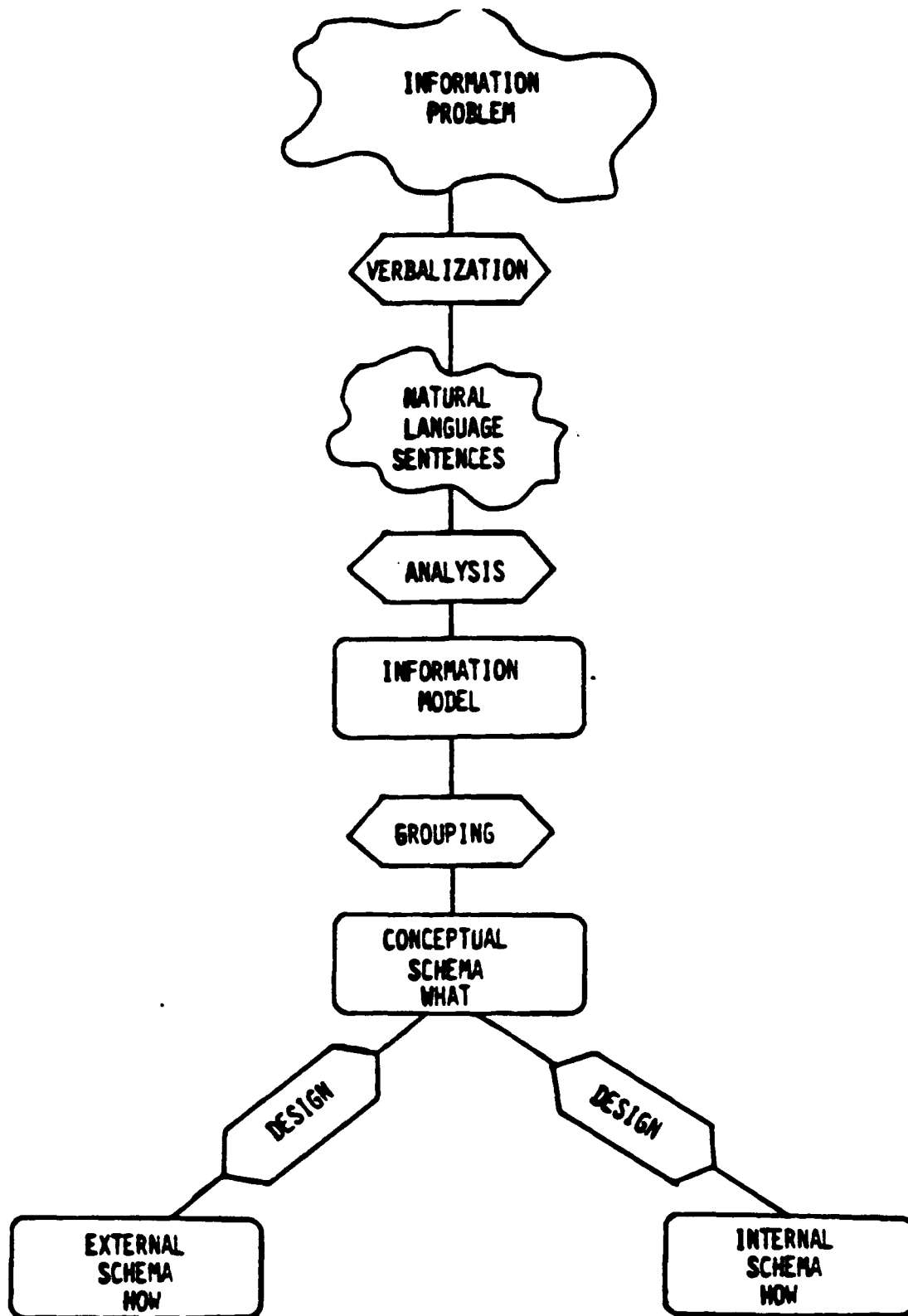
FIGURE 8. INFORMATION ANALYSIS METHODOLOGY

INFORMATION ANALYSIS (ENALIM DIAGRAM) KEY

EVERY "A" HAS ONE AND ONLY ONE "B".
A "B" IS OF ZERO, ONE OR MANY "A".

AN "A" HAS ZERO OR ONLY ONE "B".
A "B" IS OF ZERO, ONE OR MANY "A".
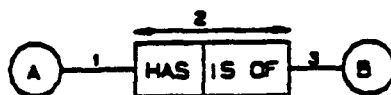
AN "A" HAS ZERO, ONE OR MANY "B".
A "B" IS OF ZERO, ONE OR MANY "A".
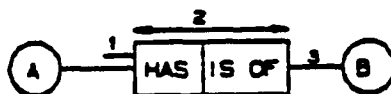
EVERY "A" HAS ONE OR MANY "B".
A "B" IS OF ZERO, ONE OR MANY "A".
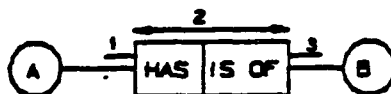
EVERY "A" HAS ONE OR MANY "B".
EVERY "B" IS OF ONE OR MANY "A".
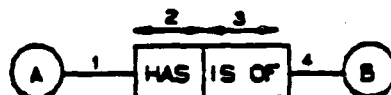
AN "A" HAS ZERO OR ONLY ONE "B".
A "B" IS OF ZERO OR ONLY ONE "A".
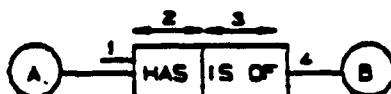
EVERY "A" HAS ONE AND ONLY ONE "B".
A "B" IS OF ZERO OR ONLY ONE "A".

EVERY "A" HAS ONE AND ONLY ONE "B".
EVERY "B" IS OF ONE AND ONLY ONE "A".
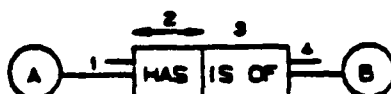
EVERY "A" HAS ONE AND ONLY ONE "B".
EVERY "B" IS OF ONE OR MANY "A".
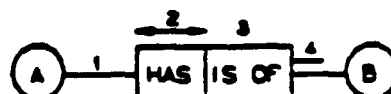
AN "A" HAS ZERO OR ONLY ONE "B".
EVERY "B" IS OF ONE OR MANY "A".

NOTE: "A" MEANS "A MEMBER OF SET A" AND "B" MEANS "A MEMBER OF SET B".

(A) A REPRESENTS A SET OF NONLEXICAL (NON-
     UTTERABLE) OBJECTS.

(B) B REPRESENTS A SET OF LEXICAL
     (UTTERABLE) OBJECTS.

(A)
|R2| BRIDGE WHICH RELATES A NONLEXICAL OBJECT
|R1| TO A LEXICAL OBJECT THROUGH ROLES R1 AND R2.
(B)

(A)
|R2| IDEA WHICH RELATES TWO NONLEXICAL OBJECTS
|R1| THROUGH ROLES R1 AND R2.
(B)

[A] THE OBJECT "A" OCCURS ELSEWHERE
     ON THIS OR ANOTHER IA DIAGRAM.
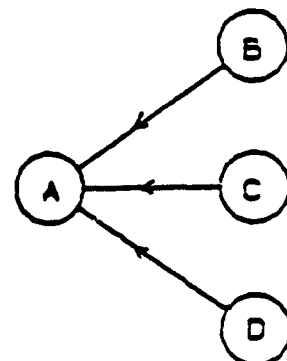
B, C AND D ARE SUBSETS OF A. A MEMBER
OF A CAN BE A MEMBER OF B, C OR D (OR
ANY COMBINATION OF B, C AND D).
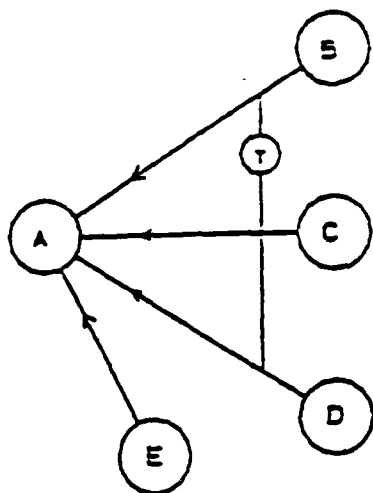
TOTAL SUBSET CONSTRAINT.

ALL OF THE MEMBERS OF A ARE CONTAINED
IN SUBSETS B, C OR D. IT MAY ALSO BE
CONTAINED IN MORE THAN ONE SUBSET, AND
COULD ALSO OCCUR IN SUBSET E.
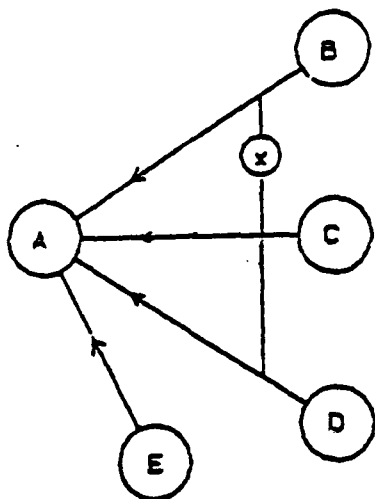
EXCLUSION SUBSET CONSTRAINT.

A MEMBER OF A IS CONTAINED IN AT MOST
ONE OF THE SUBSETS B, C OR D. IT DOES
NOT HAVE TO BE IN ANY OF THEM AND COULD
ALSO BE CONTAINED IN SUBSET E.
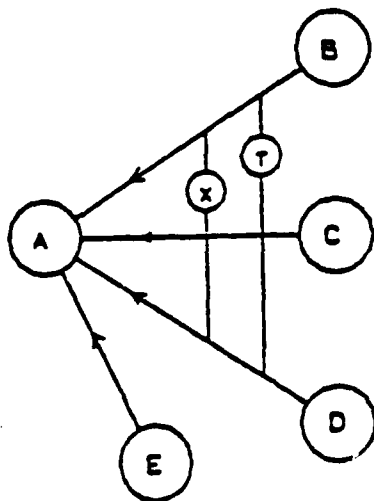
TOTAL-EXCLUSION SUBSET CONSTRAINT.

A MEMBER OF A MUST BE IN ONE AND ONLY
ONE OF THE SUBSETS B, C OR D. THIS
DOES NOT MEAN THAT IT CANNOT ALSO BE
CONTAINED IN SUBSET E.
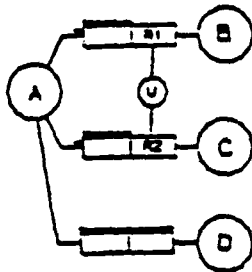
JOINT UNIQUENESS CONSTRAINT.

A MEMBER OF B PARTICIPATING IN ROLE R1 IN COMBINATION WITH A MEMBER OF C PARTICIPATING IN ROLE R2 UNIQUELY IDENTIFIES A MEMBER OF A.
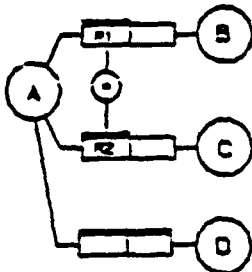
JOINT EQUALITY CONSTRAINT.

ALL MEMBERS OF A THAT ARE RELATED TO B THROUGH ROLE R1 ALSO ARE RELATED TO C THROUGH ROLE R2 AND VICE VERSA.
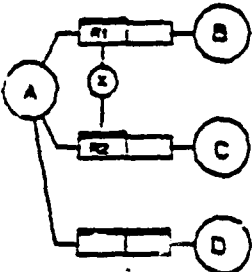
JOINT EXCLUSION CONSTRAINT.

ANY MEMBER OF A THAT IS RELATED TO B THROUGH ROLE R1 CANNOT BE RELATED TO C THROUGH ROLE R2.
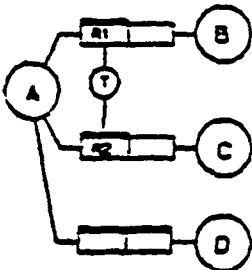
JOINT TOTAL CONSTRAINT.

A MEMBER OF A MUST PARTICIPATE IN EITHER B THROUGH ROLE R1 OR IN C THROUGH ROLE R2 OR BOTH.
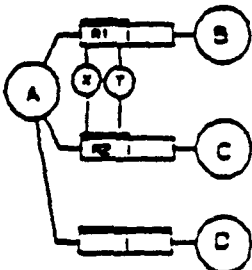
JOINT TOTAL-EXCLUSION CONSTRAINT.

A MEMBER OF A MUST PARTICIPATE IN EITHER B THROUGH ROLE R1 OR IN C THROUGH ROLE R2 BUT NOT IN BOTH.

Appendix D3

<u>Data Specification Language (DSL)</u>

AUTHOR'S NOTES

This issue of the *Data Specification Language* replaces the draft dated 01/27/86. The changes are mainly additions to the text which explains how the language is used. Concentrating on Part III, here are some items that might be of some interest:

III-3.3 defines lexical substitution which is patterned after the 'C' Language implementation.

III-4 gives some detail about the schema block.

III-5 gives some examples of entities. The association entity has been removed, but the capability is implemented differently.

III-8 expands on the idea of semantic types. This is new for this issue.

III-9 talks to static constraints. This also is new for this issue.

There is now a table of contents for each section, but the page numbers have not been filled in. I am waiting for the document to become more stable before doing this since it is a manual operation on the equipment I have been using.

Some planned additions are: a discussion of the differences and similarities between DSL and Entity-Relationship modeling and Binary-Relationship modeling (that should be interesting); and more annotated examples. The part about mapping to different physical implementations will probably resurface as an appendix sooner or later too.

Comments on this work may be directed to:

Douglas Schenck
McDonnell Douglas Aerospace Information Services Company
MDAISC
P.O. Box 516
Dept. W315
Building 271D
St. Louis, MO 63166

(314) 234-5258

==

Table of Contents

1. Introduction

The Data Specification Language (DSL) is a declarative language used to express an information model. This manual explains the principles of information modeling and how DSL is used to create models of information that are useful in understanding how an operation functions.

Part I explains the information modeling process; Part II is a formal definition of DSL in terms of (a form of) BNF; Part III is a guide to the use of DSL; Part IV is a glossary of the terms used throughout the manual.

The aim of information modeling is to capture the essence of the information needed for the operation of some enterprise. Its aim is not to model data, i.e., to model the form of information as it might be used in an application environment. It is important, however, that the information model be able to produce such a data model. In fact, the production of data models[1] is the second most important role of the information model.

This begs the question - what is the most important role of the information model? It is simply this - to express what is known about enterprise information. This expression is aimed equally at human users and computing machines.

The purpose of the information processing system is to encourage the use and exchange of information. Both the provider and the receiver of data should be able to understand it in exactly the same manner. i.e., to treat data as information. The use of an information model is aimed at avoiding errors in interpreting the meaning of data.

Data and information, as used in this document, have the following meanings:

-- Data is the representation of information independent of its meaning. e.g., an integer number.

-- Information is the data plus the meaning connected with it. It is any kind of knowledge about things or concepts that is exchangeable among users. e.g., an integer number is the age, in years, of some person.

The Data Specification Language (DSL) enables the expression of the information needs of an enterprise. Its purpose is to

* * * * *

[1] You will notice that information model is always treated as a singular thing while data model is treated in a plural sense. This is because there should be only one information model, but several data models that implement the information model are possible.

capture the meaning of data so that the provider and receiver of data may interpret it without error. DSL is not a database design tool, although it is possible to derive a database design from the information model written in DSL.

2. The Information Model

The information model establishes the framework within which information may be expressed. Not all kinds of information; only the information that can be represented by an information processing system. In general this means data, with meaning, that has a realized form and predictable structures and relationships. It is not possible, nor is it necessary, to represent all kinds of information about things. When modeling a person for example, there are all kinds of qualities that defy a meaningful representation[2]; how does one feel about something or what does one believe about some topic. Fortunately, information processing systems usually deal only with concrete facts such as a person's name or age, or with simulations of abstract feelings (do you strongly agree, agree, strongly disagree, etc. with this or that statement).

The information model is concerned with the expression of things (and concepts) needed by (for the operation of) the enterprise of interest. The concepts needed to accomplish that expression are briefly described here and will be discussed in more detail in subsequent sections.

An entity is the abstraction of some artifact or concept of interest and use to the enterprise.

An attribute is a defining fact about an entity.

A class is a collection of entities that are (thought to be) alike in some manner.

A rule is a constraint which must be enforced by the information processing system. e.g., an employee must belong to exactly one department, or all points must be in the first octant. Rules may vary somewhat among different, but similar, enterprises. Most, but perhaps not all, businesses will be in accord with the first example. Few geometric modellers would agree with the second example.

An operation defines how an entity may be used within the enterprise and what the result of the operation may be. e.g., a number may be used in an add operation and the result is a number.

* * * * *

[2] Artificial Intellegence (AI) and advancing computing technology may someday alter this viewpoint.

A _function_ is an algorithm which yields some result object.

A _procedure_ is an algorithm which operates on information to produce some desired end state.

The _relationship_ is a mechanism used to connect two entities in some fashion. e.g., a Line uses a Point as one of its attributes. That would be a defining relationship.

The _schema_[3] is what we call the the information model. It embodies what we (think we) know about enterprise information.

3. Data Models

There may be a number of data models within an enterprise that implement (part of) the information model. It is simply impossible to cover all of the possible data models that might be employed. Even when talking about a generic data model (e.g., relational) we cannot cover all of the possible implementation variations.

However, the purpose of this section is not to discuss specific data models, but to explain how data models are different from the information model. Consider this parallel: we have a problem for which there are many possible solutions. Some process of elimination will yield a single solution that seems to be the best according to some criteria. When separate groups attempt to solve the same problem we might expect to find that each group produces a different solution. Very possibly each group would evaluate the alternatives based on differing criteria.

The same thing happens with the translation from the information model[4] to a data model. Group A and Group B look at the information model and come up with a data implementation that best suits their needs and environment. The best solution for Group A may not be the best for Group B. This does not become a problem until Group A wants to share information with Group B.

Now we need the information model to stand between the separate implementations to interpret the meaning of data found at either side of the exchange. This works best when the information model is formal rather than ad hoc.

* * * * *

[3] Not to be confused with the term _conceptual schema_ as used by the ANSI/X3/SPARC DBSG.

* * * * *

[4] We assume that all are looking at exactly the same information model and all understand it exactly the same way. When the basic understanding is different, effective communication is impossible.

So, the data models are concerned with application needs and the implementation environment. The information model must not be concerned with these issues. The information model must express concepts in a common language. The data model does not have to do this (but it doesn't hurt if it does).

4.   Elements of the Information Model

A brief description of the elements of the information model was given earlier. This section explains them in complete detail.

4.1 Entity

An entity is the abstraction of some artifact or concept of interest and use to the enterprise. An entity must posess these qualities:

-- it must express an entire concept

-- it must exist independently

This means that an entity must not express a partial concept. By way of contrast, data models often express partial concepts in physical structures.   To exist independently means that an entity must not depend on any other entity for its existance[5].   Again, data models often rely on dependent relationships, especially where partial concepts are required.   An example will help to illustrate these ideas.

We wish to define a FairCurve, which is a smooth curve that passes through two or more points. The behaviour of the curve at each point is defined by attributes telling whether or not slope and curvature continuity is required. The very first and last points have no such attributes. Furthermore, there is an integer number attribute of the FairCurve that somehow determines the method of interpolation between points.

One way of building a data model is with the following relational tables (there are other ways of doing this):

* * * * *

[5] There is the possibility that a particular entity may be treated as "dependent with respect to" another entity. See *dependent* in 4.8.

FAIRCURVE

<-------->

ID	INTERP	FIRST	LAST
========	========	========	========
FC0001	3	PT1234	PT3456
FC0002	4	PT0123	PT5600
etc.			

FAIRCURVE-POINT

<--------------->

| ID | SEQ | PTREF | G1 | G2 |
|========|=====|========|=======|=======|
| FC0001 | 003 | PT7164 | TRUE | FALSE | [6]
FC0002	001	PT4617	FALSE	FALSE
FC0002	003	PT4614	TRUE	FALSE
FC0002	002	PT7117	FALSE	FALSE
FC0001	002	PT4314	TRUE	TRUE
FC0001	001	PT4143	FALSE	FALSE
FC0001	004	PT4341	FALSE	FALSE
FC0002	004	PT4334	TRUE	FALSE
FC0002	006	PT4146	TRUE	TRUE
FC0002	005	PT4943	FALSE	FALSE
etc.				

First, it is clear that the records in FairCurvePoint are
dependent on the FairCurve table. i.e., if a record is removed
from FairCurve, then all corresponding records in FairCurvePoint
must be removed also. This is a clue that FairCurvePoint is
really an attribute of FairCurve. But more on this later.

It is not so clear whether or not a record in FairCurvePoint
can be deleted without affecting the definition of a given
FairCurve.

Second, does FairCurvePoint express a complete idea? In
other words, could we explain what this relation means without
calling up the context in which it is used. The answer to this is
an unqualified maybe, but the claim here is that FairCurvePoint
cannot be explained without bringing FairCurve into the
discussion also.

We can conclude from this that FairCurvePoint is not an
entity, at least in the context of the information model. Neither
is FairCurve, since by itself it is not complete. They may be
called entities in the jargon of the particular data model that

* * * * *

[6] Notice that the combination of ID and SEQ must be unique. SEQ
    gives the ordering of points interior to the FairCurve.
    Moreover, in relational tables there is no guarantee that
    "key" values will be in any meaningful order. They are shown
    in random order to emphasize this consideration.

is being employed. A solution to this problem, in information
modeling terms, will be given later.

4.2 Attribute

An attribute is a particular fact about an entity, such as x
is a fact about Point. Each entity will have a set of attributes
that make it unique from all other entities. The name of the
entity is shorthand for the set of facts needed to define it.

Attributes may be classified according to the purpose they
play in the description of an entity. The terms used are: Role
and NoRole. A purely conceptual schema will employ only Role
attributes in the description of entities. NoRole attributes will
be used as the schema migrates toward a physical implementation.

A Role attribute is essential for the description of an
entity. If one or more of the Role attributes were eliminated
from the description, the meaning of the entity would be lost.

A NoRole attribute, on the other hand, is not essential for
the understanding of the entity in question. A NoRole attribute
is (or may be) necessary for the function of an operational
system.

One common example of a NoRole attribute is the *key* attribute
which is used to gain access to a specific entity. These key
attributes are, more often than not, manufactured. The data
representation of the key attribute is likely to be *different* for
different implementations. e.g., a relational database would
probably use a character string for key values while a network
database would probably use memory addresses or record pointers
for that purpose.

In either event the NoRole attributes may be ignored for the
purpose of understanding an entity; only the Role attributes are
essential for that purpose.

An attribute may be viewed as having three components: the
name of the attribute; its semantic type; and any constraints
that might be enforced on particular values for its semantic
type. Constraints may or may not be present, and if not the
semantic type is unconstrained.

The name of an attribute is known only to the entity in which
it is defined. i.e., if two entities have the same attribute
(name) there is no inference that the two attributes mean the
same thing.

The semantic type defines what the attribute is. It does not
define how the attribute is to be represented in a specific
environment. One should not assume that the semantic type *Integer*
will be realized as a binary twos complement number. It could
also be encoded as a packed or unpacked decimal, or as a
character string (of digits).

Even though a semantic type and a data type are used in different roles, it is difficult to separate these concepts in practice.

The semantic types built into DSL are listed below. Part III explains how the semantic types are used.

SIMPLE
   -- Number
      -- Integer
      -- Real
         -- Float
         -- Fixed
  -- String
  -- Logical

RELATIONSHIP
  -- Refer

GROUPING
   -- Array
   -- List
   -- Set

EXTENSIONAL
  -- Defined

COMPLEX
   -- Enumeration
   -- Structure

Note that the complex semantic types (enumeration and structure) may not be used directly as semantic types. They must first be given a semantic type name by the use of the defined semantic type.

4.3 Class

A class is a collection of things that are (thought to be) alike in some manner. Unlike entity, which has some kind of implementation, the class is strictly abstract. There are no instances of a class in a database environment.

A class may contain either classes or entities as members. When a class is a member of a class it is called a sub-class. Sub-classes can be used to define a hierarchial structure such as:

```
Geometry
  Point
  Curve
     circle
     line
     ellipse
     etc.
  Surface
     plane
     cylinder
     cone
     etc.
```

A member of a class can also be a member of another class. This allows an entity to be organized in different ways for different purposes. e.g.,

```
Animal
   wolf
   dog
   cat
Canine
   wolf
   dog
   fox
Pet
   dog
   cat
```

Dog is in the classes animal, canine, and pet. Wolf is a canine and an animal, but not a pet (well, lets hope not).

The purpose of a class in a conceptual schema is to organize and simplify. We could define a class called Geometry with its members being the entities that are used to represent position and shape. Then, whenever we need something that represents position and shape we would refer to Geometry instead.

4.4 Rule

A rule is an expression of a constraint that is to be enforced in some manner when an entity instance is created or modified, or possibly, when it is deleted. A rule might be enforced by a DBMS, application code, manual procedure, or by other means. The expression of a rule in the information model does not mean that the responsibility of enforcement falls on the DBMS.

Rules can be simple or complex. A simple rule might state that an attribute can only have the values zero through ten. If we look at the FairCurve example given earlier we can observe a more complex constraint; one involving two attributes. The table containing G1 and G2 has either True or False values. However, when the value of G1 is False then the value of G2 must also be False.

More examples of rules can be found in the User's Guide,
Part III of this manual.

4.5 Operation

Each type of data will have some (possibly open ended) set of
operations that may be performed on it. Each operation will
yield some well defined result; e.g., Curves may be intersected
yielding a point, dates may be subtracted yielding an age,
vectors may be crossed yielding a vector, numbers may be added
yielding a number, and so on. Presumably, if no operations are
defined for a given type, there is no need for that type.

A significant part of the discovery process should be
concerned with the identification of the operation set that
applies to the data. Without that information the information
model is incomplete.

It is interesting to note that in mathematics, operations on
numbers, sets, vectors, etc. are rigorously defined and are in
place as part of the discipline. This account in part for the
ability of mathematicians to communicate well in unambiguous
terms with one another. No less should be expected from other
"disciplines".

4.6 Function

A function is a procedure that yields some object as a
result. That object may be simple (e.g., a number) or complex
(e.g., a point entity). A function can be compared to an
operation. The difference lies in the way it is invoked and in
the manner in which inputs are presented.

e.g., an operation is stated:

object operation-symbol object   5 + 6

where a function is invoked by:

function-name(parameter-list)   add(5,6)

In this case the result is exactly the same (the number 11).
The decision that selects either a function or an operation to
produce some result is based partly on convention and aesthetics.

4.7 Procedure

A procedure is a collection of expressions that produces a
desired end state. Rules, operations, and functions are forms of
procedures. Procedures may have parameters that provide the
variable data on which operations are performed.

4.8 Relationship

A relationship relates two entities in a particular manner. A relationship can be viewed as a special kind of entity with the following attributes:

CARDINALITY defines the ratio of things that are being related to one another and is specified by the Array, List, or Set semantic types, or the absence of any one of these.

Cardinality applies to relationships through the Refer semantic type and to the other non-relational semantic types. e.g.,

```
entity a;
role
  b : array(10) of real;
  c : array(10) of refer(d);
  e : real;
  f : refer(g);
end;
```

for each a there are 10 b's; for each a there are 10 c's; for each a there is one e; and for each a there is one f. In the case of the c attribute a relationship between a and d is established. In the case of the f attribute a relationship is established to g.

For non-relational semantic types the following reverse cardinality holds (using a for the entity and b for the attribute):

for each a there are n b's

for each b there is one a.

For the relational semantic type (refer) the following reverse relationship holds:

for each a there are n b's

for each b there may be n a's.

That is, because entities are independent, they may be freely referenced by any other entity. This has the effect of yielding a many-to-many relationship whenever a refer semantic type is employed.

REQUIREDNESS states whether a particular attribute is manditory or optional. Optional attributes are sometimes meaningful; e.g., in the definition of a Face the Cutouts are optional. The use of optional has the effect of producing a zero-to-n relationship.

DEPENDENCE states whether or not the referenced entity of the relationship owes its existance to the refering entity. This is meaningful only when used with the refer semantic type. When DEPENDENT is used it states that the referenced entity exists (in this context) only for the purpose of defining the referring entity. In effect it treats the referenced entity the same as a non-relational semantic type.

PURPOSE states whether an attribute provides a fact (a Role purpose) or ancilary data (a NoRole purpose) about an entity. It is not common in information models to provide NoRole attributes. They are often used in data models.

4.9 Schema

The schema embodies all of the separate constructs discussed before as an information model. This should not be confused with the terms conceptual schema, external schema, and internal schema as used by the ANSI/X3/SPARC DBSG. Schema is used simply as shorthand for information model.

5. The n-Schema Framework

The ANSI/X3/SPARC DBSG<1> gives us a basic framework for a three schema architecture comprised of conceptual, external, and internal schema. This architecture assumes that a database management system (of some kind) will be employed by the information processing system.

The conceptual schema comprises a central description of the data that may be in a database. This conceptual schema may be viewed in a number of different ways by users of the data. These views are called the external schema. The view of the data as it is stored in a file is called an internal schema.

This architecture conveniently permits different viewpoints to be expressed without interfering with the requirements of others and works well in a homogeneous information processing system environment.

We can diagram this architecture as:

```
                    [CONCEPTUAL]
                         |
             +-------+-------+
             |               |
            <p>             <p>
         [EXTERNAL]     [INTERNAL]
```
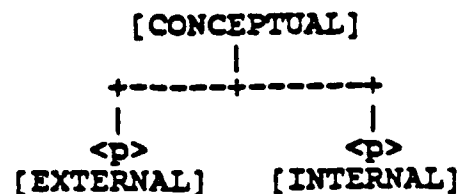
(<p> denotes a "to-many" relationship)

A fourth schema (the information model) can be introduced into this framework to account for a heterogeneous information processing system environment.

```
            [INFORMATION MODEL]
                    |
                   <p>
              [CONCEPTUAL]
                    |
          +--------+--------+
          |                 |
         <p>               <p>
      [EXTERNAL]        [INTERNAL]
```
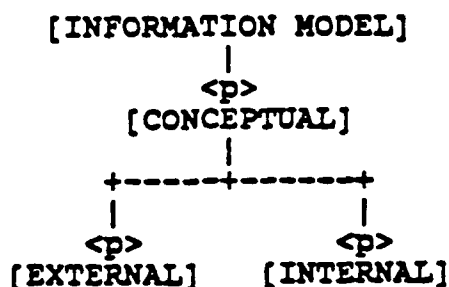
That is, there may be many conceptual schema, each one expressed in the terms of the particular database environment. Each of these conceptual schema are derived from the information model of which there is only one.

6. The Discovery Process

The information model acts as a bridge between the discovery of data and the eventual implementation of it. The discovery phase consists of the identification of information and how it is used within the enterprise. Several existing methodologies might be employed during this discovery phase. One important aspect of this discovery phase is the identification of the operations that may be performed upon information, the results of operations, and the behaviour of information while the operations are being performed.

The implementation phase takes into account the environment, which includes the specific computer, operating system, database management system, development language, performance needs, and many other considerations and constraints.

Mapping functions are used between these three phases. The mapping function between discovery and the information model is probably manual in nature. It is possible to automate the mapping between the information model and the implementation given that the information model is sufficiently complete.

```
                    PHASE
          *------------------*
          | DISCOVERY        |------*
          *------------------*      |
     *---------<MAP>--------------------*
     |       *------------------*
     *---->| INFO MODEL        |------*
          *------------------*        |
     *---------<MAP>--------------------*
     |       *------------------*
     *---->| IMPLEMENTATION |
          *------------------*
```
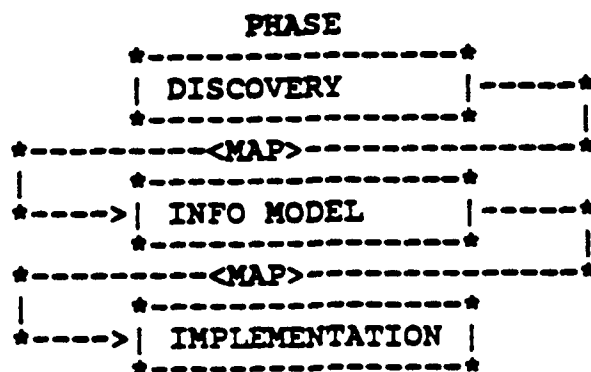
7. Requirements for the Information Model

The information model must fulfill a number of basic require-
ments. In addition, it is desireable that it be able to support a
number of peripheral activities related to the process of
creating, maintaining, and documenting the information model.

Basic Requirements

Describe the artifacts needed for the operation of a given
enterprise. Mainly this means accounting for the entities,
their names, attributes, and semantic definitions.

Define the operations that are permissible for each entity
(and class where appropriate). This also includes the
definition of any preconditions and assumptions connected
with the performance of an operation (e.g., the two
vectors in a cross product must not be parallel), the
possible error conditions (e.g., what if two curves don't
intersect), and the result type of those operations.

Define the way entities are organized into classes which
may be treated as a whole for certain purposes.

Define the constraints that must be enforced on single or
complex values.

Accomplish this in a way that is usable by both people and
computing equipment.

Secondary Requirements

Capture the information model as an encoded computer model
(not just a text file).

Produce from this encoded computer model a variety of
presentations aimed at different people and activities.
Some examples of these presentations are: entity
relationship diagrams, binary relationship diagrams,
natural language (English, German, French, etc.)
narriative of the content of the information model, cross
references, and so on.

Automatic production of end user documentation. This has
been separated from the paragraph above to emphasize its
importance.

Production of reports to isolate differences between
versions of the information model.

Production of secondary forms of the information model.
e.g., database definitions, procedural code, data
declarations, exchange file formats, etc.

8. The Role of the Information Model

The information model is a central, unique, and neutral expression of the information needed by some enterprise. It is independent of the form actually taken by an implementation. It makes visible to both human users and computers the content, structure, and most importantly, the meaning of the data described by it; albeit meaning probably has a different flavor to humans and computers.

Since the information model might be implemented in different ways by different systems, a way of coordinating those differences is needed. These are differences of form and not meaning. The information model provides this coordination, along with mapping schema, which at this time must be expressed in procedural terms.

For example, say the information model defines line as a point and a vector, but that some foreign CAD system defines it as two points. The mapping between the different forms is accomplished by a collection of procedures called a translator. The translator must convert the intent of the foreign format to the intent of the information model. These conversions are sometimes trivial, but they can be vexingly difficult also. One important consideration while building an information model is to make conversion (as) easy (as possible).

The roles of the information model are summarized below:

1. To provide a common basis for understanding the characteristics and behaviour of the information for the enterprise of interest.

2. To provide a basis for the realization of the physical form which represents these information.

3. To provide a basis for the mapping between different physical

The enterprise determines the requirements for what is in the information model and how it is expressed. Since *STEP* is an exchange enterprise the focus is on the transfer of meaning, rather than on computational efficiency. The external view (i.e., the way a particular CAD/CAM system sees an entity) of the same data may be very much different since its priorities are probably different. The external system may choose to represent geometry by coefficients to agree with particular algorithms and to improve computational efficiency. Each external environment will have its own viewpoint about what is needed to support its operational requirements.

Alphabetical Index to Elements

```
          II-00  ........  <FORMAL-PARAMETER-LIST>
          II-00  ........  <FORMAL-PARAMETER>
          II-00  ........  <FUNC-BODY>
          II-00  ........  <FUNC-DECL>
          II-00  ........  <FUNC-HEADER>
(Basic)   II-00  ........  <FUNC-ID>
          II-00  ........  <FUNCTION-CALL>
(Basic)   II-00  ........  <ID-CHAR>
          II-00  ........  <IDENTIFIER-LIST>
(Basic)   II-00  ........  <IDENTIFIER>
          II-00  ........  <IF-STATEMENT>
          II-00  ........  <IN-CLAUSE>
          II-00  ........  <INCREMENT-CONTROL>
          II-00  ........  <INT-LIT-ID>
(Basic)   II-00  ........  <INTEGER-LITERAL>
          II-00  ........  <INTEGER-TYPE>
(Basic)   II-00  ........  <LETTER>
          II-00  ........  <LEXICAL-CONSTANT-BODY>
          II-00  ........  <LEXICAL-CONSTANT-DECL>
          II-00  ........  <LEXICAL-CONSTANT-HEADER>
          II-00  ........  <LEXICAL-CONSTANT>
(Basic)   II-00  ........  <LEXICAL-ID>
          II-00  ........  <LIKE-CLAUSE>
          II-00  ........  <LIKE-STRING>
          II-00  ........  <LIMIT-SPEC>
          II-00  ........  <LIMIT>
          II-00  ........  <LIST-TYPE>
          II-00  ........  <LIT-ID>
          II-00  ........  <LITERAL-LIST>
          II-00  ........  <LITERAL>
          II-00  ........  <LOCAL-BODY>
          II-00  ........  <LOCAL-DECL>
          II-00  ........  <LOCAL-HEADER>
(Basic)   II-00  ........  <LOCAL-ID>
          II-00  ........  <LOGICAL-CLAUSE>
(Basic)   II-00  ........  <LOGICAL-LITERAL>
          II-00  ........  <LOGICAL-OP>
          II-00  ........  <LOGICAL-TYPE>
(Basic)   II-00  ........  <LOWER>
          II-00  ........  <NEST>
(Basic)   II-00  ........  <NEWLINE>
          II-00  ........  <NULL-CLAUSE>
          II-00  ........  <OPER-DECL>
(Basic)   II-00  ........  <OPER-ID>
          II-00  ........  <OPERATION-BODY>
          II-00  ........  <OPERATION-HEADER>
          II-00  ........  <OPERATOR>
          II-00  ........  <PARAMETER-LIST>
          II-00  ........  <PARAMETER>
          II-00  ........  <PRECISION-SPEC>
          II-00  ........  <PROC-BODY>
          II-00  ........  <PROC-CALL-STATEMENT>
          II-00  ........  <PROC-DECL>
          II-00  ........  <PROC-HEADER>
(Basic)   II-00  ........  <PROC-ID>
```

```
              II-00 ........ <QUALIFIED-SEMANTIC-TYPE>
     (Basic)  II-00 ........ <QUOTE>
              II-00 ........ <REFER-TYPE>
     (Basic)  II-00 ........ <REMARK>
              II-00 ........ <REPEAT-CONTROL>
              II-00 ........ <REPEAT-STATEMENT>
     (Basic)  II-00 ........ <REPLACE-ID>
              II-00 ........ <RESULT-TYPE>
              II-00 ....... <RULE-BODY>
              II-00 ........ <RULE-CALL>
              II-00 ........ <RULE-DECL>
              II-00 ........ <RULE-HEADER>
     (Basic)  II-00 ........ <RULE-ID>
              II-00 ........ <SCHEMA-BODY>
              II-00 ........ <SCHEMA-DECL>   *** ROOT CONSTRUCT ***
              II-00 ........ <SCHEMA-HEADER>
     (Basic)  II-00 ........ <SCHEMA-ID>
              II-00 ........ <SCHEMA-INFO-KEYWD>
              II-00 ........ <SCHEMA-INFO>
              II-00 ........ <SEMANTIC-TYPE>
     (Basic)  II-00 ........ <SIGN>
              II-00 ....... <SKIP-STATEMENT>
     (Basic)  II-00 ....... <SPACE>
     (Basic)  II-00 ....... <SPECIAL-CHAR>
              II-00 ....... <STATEMENT>
     (Basic)  II-00 ....... <STRING-LITERAL>
              II-00 ....... <STRING-OP>
              II-00 ....... <STRING-TYPE>
              II-00 ....... <STRUCTURE-TYPE>
     (Basic)  II-00 ....... <TAB>
     (Basic)  II-00 ....... <TAIL-REMARK>
     (Basic)  II-00 ....... <TYPE-ID>
              II-00 ....... <TYPE-SPEC>
     (Basic)  II-00 ....... <UNSIGNED-NUMBER>
              II-00 ....... <UNTIL-CONTROL>
     (Basic)  II-00 ....... <UPPER>
              II-00 ....... <WHERE-CLAUSE>
              II-00 ....... <WHILE-CONTROL>
     (Basic)  II-00 ....... <WHITE-SPACE>
              II-00 ....... <WITH-STATEMENT>
```

1. Notation

A kind of BNF is used to present the syntax of the Data Specification Language. The notation conventions are given below. These syntax definitions are given with a minimum of commentary. Part-III explains usage.

1. A word enclosed in angle brackets is an element of the language. The word inside the brackets is the name of the element.

   e.g., <character>

2. A literal is displayed as *italic* characters. All literals are written exactly as shown, except either upper or lower case letters may be used.

3. The vertical bar is the "or" operator, which means that a choice is to be made from the elements separated by it.

   e.g., <letter> | <digit>

4. An element enclosed in square brackets is optional, that is, it may occur zero or one times.

   e.g., [<optional>]

5. An element with an ellipsis suffix may be repeated many times.

   e.g., <one-or-more>...
   e.g., [<zero-or-more>]...

8. The symbol ::= indicates a production. The element on the left is defined to be the elements on the right.

   e.g., <identifier> ::= <letter> [<id-char>]...

9. A remark in the syntax descriptions is designated by >> and the text following on the same line. This is similar to the tail remark of the language.

10. Parentheses are used in the grammar as necessary to avoid ambiguity.

11. The grammar rules are divided into three sections and are listed in alphabetical order within each section.

```
<ACTION-BODY> ::=
    [<BLOCK-DECL>]
    [<LOCAL-DECL>]
    <COMPOUND-STATEMENT>


<ARITH-OP> ::=
    <ADD-OP> |
    <SUB-OP> |
    <MULT-OP> |
    <REAL-DIV-OP> |
    <EXP-OP> |
    <INT-DIV-OP> |
    <MOD-OP>


<ARRAY-TYPE> ::=
    array <BOUND-SPEC> of <QUALIFIED-SEMANTIC-TYPE>


<ASSIGNMENT-STATEMENT> ::=
    <IDENTIFIER> = <EXPRESSION> ;


<ATTRIBUTE-BODY> ::=
    <ATTRIBUTE-DECL-STMT>...


<ATTRIBUTE-DECL-STMT> ::=
    <IDENTIFIER-LIST> : [optional] [dependent]
    <QUALIFIED-SEMANTIC-TYPE> [<WHERE-CLAUSE>] ;


<ATTRIBUTE-DECL> ::=
    <ATTRIBUTE-HEADER>
    <ATTRIBUTE-BODY>


<ATTRIBUTE-HEADER> ::=
    key |
    role |
    keyrole |
    norole


<BETWEEN-CLAUSE> ::=
    <LIT-ID> between <LIT-ID> and-op <LIT-ID>


<BI-PRECISION-SPEC> ::=
    ( <INTEGER-LITERAL> , <INTEGER-LITERAL> )
```

==

```
<BLOCK-DECL> ::=
    [<BLOCK-MEMBERS>]...


<BLOCK-MEMBERS> ::=
    <DEFINED-TYPE-DECL> |
    <ENTITY-DECL> |
    <CLASS-DECL> |
    <FUNC-DECL> |
    <PROC-DECL> |
    <RULE-DECL> |
    <OPER-DECL> |
    <LEXICAL-CONSTANT-DECL>


<BOUND-SPEC> ::=
    ( <EXPRESSION> [ : <EXPRESSION> ] )


<CASE-ACTION> ::=
    <CASE-LABEL> : <STATEMENT>


<CASE-BLOCK> ::=
    <CASE-ACTION>...
    [<CASE-OTHERWISE>]
    end ;


<CASE-LABEL> ::=
    <LITERAL> [<, <LITERAL>]...


<CASE-OTHERWISE> ::=
    otherwise : <STATEMENT>


<CASE-STATEMENT> ::=
    case <IDENTIFIER> of
    <CASE-BLOCK>


<CLASS-BODY> ::=
    of <CONSTITUENT-LIST> ;


<CLASS-DECL> ::=
    <CLASS-HEADER>
    [<CLASS-BODY>]


<CLASS-HEADER> ::=
    class <CLASS-ID>
```

```
<COMP-OP> ::=
    <LESS-THAN> |
    <GREATER-THAN> |
    <EQUAL> |
    <LESS-EQUAL> |
    <GREATER-EQUAL> |
    <NOT-EQUAL>


<COMPLEX-SEMANTIC-TYPE> ::=
    <STRUCTURE-TYPE> |
    <ENUMERATION-TYPE>


<COMPOUND-STATEMENT> ::=
    begin
    [<STATEMENT>]...
    end ;


<CONDITION-SPEC> ::=
    <CONDITION> [<LOGICAL-OP> <CONDITION>]...


<CONDITION> ::=
    <LOGICAL-CLAUSE> |
    <BETWEEN-CLAUSE> |
    <IN-CLAUSE> |
    <LIKE-CLAUSE> |
    <NULL-CLAUSE>


<CONSTITUENT-LIST> ::=
    ( <CONSTITUENT> [, <CONSTITUENT>]... )


<CONSTITUENT> ::=
    <CLASS-ID> |
    <ENTITY-ID>


<DEFINED-TYPE-BODY> ::=
    <DEFINED-TYPE>...


<DEFINED-TYPE-DECL> ::=
    <DEFINED-TYPE-HEADER>
    <DEFINED-TYPE-BODY>


<DEFINED-TYPE-HEADER> ::=
    type
```

```
<DEFINED-TYPE> ::=
     <TYPE-ID> = <QUALIFIED-SEMANTIC-TYPE> [<WHERE-CLAUSE>] ;


<DUMMY-STATEMENT> ::=
     ;


<ENTITY-BODY> ::=
     <ATTRIBUTE-DECL>...
     end ;


<ENTITY-DECL> ::=
     <ENTITY-HEADER>
     [<ENTITY-BODY>]


<ENTITY-HEADER> ::=
     entity <ENTITY-ID> [<WHERE-CLAUSE>] ;


<ENUMERATION-TYPE> ::=
     enumeration of <ENUM-ID-LIST>


<ESCAPE-STATEMENT> ::=
     escape ;


<EXPRESSION-EXT> ::=
     <OPERATOR> <FACTOR>


<EXPRESSION> ::=
     [+ | - ] <FACTOR> [<EXPRESSION-EXT>]...


<FACTOR> ::=
     <IDENTIFIER> |
     <LITERAL> |
     <FUNCTION-CALL> |
     <NEST>
     <NOT-OP> <FACTOR>


<FIXED-TYPE> ::=
     fixed [ ( <PRECISION-SPEC> | <BI-PRECISION-SPEC> ) ]


<FLOAT-TYPE> ::=
     float [<PRECISION-SPEC>]
```

```
<FORMAL-PARAMETER-LIST> ::=
    ( [ <FORMAL-PARAMETER> [; <FORMAL-PARAMETER>]... ] )


<FORMAL-PARAMETER> ::=
    [var ] <TYPE-SPEC>


<FUNC-BODY> ::=
    <ACTION-BODY>


<FUNC-DECL> ::=
    <FUNC-HEADER>
    <FUNC-BODY>


<FUNC-HEADER> ::=
    function <FUNC-ID> ( [<FORMAL-PARAMETER-LIST>]
    ) : <RESULT-TYPE> ;


<FUNCTION-CALL> ::=
    <FUNC-ID> <PARAMETER-LIST>


<IDENTIFIER-LIST> ::=
    <IDENTIFIER> [, <IDENTIFIER>]...


<IF-STATEMENT> ::=
    if <CONDITION-SPEC> then <STATEMENT>
    [else <STATEMENT>]
    end ;


<IN-CLAUSE> ::=
    <LIT-ID> in-op <LITERAL-LIST>


<INCREMENT-CONTROL> ::=
    <LOCAL-ID> = <LIT-ID> to <LIT-ID> [by <LIT-ID>]


<INT-LIT-ID> ::=
    <INTEGER-LITERAL> |
    <IDENTIFIER>


<INTEGER-TYPE> ::=
    integer [<PRECISION-SPEC>]
```

==

```
<LEXICAL-CONSTANT-BODY> ::=
    <LEXICAL-CONSTANT>...


<LEXICAL-CONSTANT-DECL> ::=
    <LEXICAL-CONSTANT-HEADER>
    <LEXICAL-CONSTANT-BODY>


<LEXICAL-CONSTANT-HEADER> ::=
    define


<LEXICAL-CONSTANT> ::=
    <LEXICAL-ID> = <LEXICAL-CONSTANT> ;

    >> Note that the semicolon character may not be part of <<
    >> the lexical-constant as it is used as a terminator   <<


<LIKE-CLAUSE> ::=
    <LIT-ID> like-op <LIKE-STRING>


<LIKE-STRING> ::=
    <STRING-LITERAL> |
    <IDENTIFIER>


<LIMIT-SPEC> ::=
    ( <LIMIT> )


<LIMIT> ::=
    <EXPRESSION> to <EXPRESSION> | ?

>> ? is used to denote an indefinite upper bound <<


<LIST-TYPE> ::=
    list <LIMIT-SPEC> of <QUALIFIED-SEMANTIC-TYPE>


<LIT-ID> ::=
    <LITERAL> |
    <IDENTIFIER>


<LITERAL-LIST> ::=
    ( <LITERAL> [, <LITERAL>]... )
```

```
<LITERAL> ::=
    <FLOAT-LITERAL> |
    <FIXED-LITERAL> |
    <INTEGER-LITERAL> |
    <STRING-LITERAL> |
    <LOGICAL-LITERAL>


<LOCAL-BODY> ::=
    <ATTRIBUTE-DECL-STMT>... ;


<LOCAL-DECL> ::=
    <LOCAL-HEADER>
    <LOCAL-BODY>


<LOCAL-HEADER> ::=
    local


<LOGICAL-CLAUSE> ::=
    <EXPRESSION> [<COMP-OP> <EXPRESSION>]


<LOGICAL-OP> ::=
    <AND-OP> |
    <OR-OP>


<LOGICAL-TYPE> ::=
    logical


<NEST> ::=
    ( <EXPRESSION> )


<NULL-CLAUSE> ::=
    <IDENTIFIER> is null


<OPER-DECL> ::=
    <OPERATION-HEADER>
    [<OPERATION-BODY>]


<OPERATION-BODY> ::=
    <ACTION-BODY>
```

==

```
<OPERATION-HEADER> ::=
    operation <IDENTIFIER>
    ( [<IDENTIFIER> ,] <STRING-LITERAL> <IDENTIFIER>  )
    : <QUALIFIED-SEMANTIC-TYPE> ;


<OPERATOR> ::=
    <ARITH-OP> |
    <STRING-OP> |
    <LOGICAL-OP> |
    <COMP-OP>


<PARAMETER-LIST> ::=
    ( [ <PARAMETER> [, <PARAMETER>]... ] )


<PARAMETER> ::=
    <FACTOR>


<PRECISION-SPEC> ::=
    ( <INTEGER-LITERAL> )


<PROC-BODY> ::=
    <ACTION-BODY>


<PROC-CALL-STATEMENT> ::=
    <PROC-ID> <PARAMETER-LIST> ;


<PROC-DECL> ::=
    <PROC-HEADER>
    <PROC-BODY>


<PROC-HEADER> ::=
    procedure <PROC-ID> ( [<FORMAL-PARAMETER-LIST>] ) ;


<QUALIFIED-SEMANTIC-TYPE> ::=
    <FIXED-TYPE> |
    <ARRAY-TYPE> |
    <LIST-TYPE> |
    <REFER-TYPE> |
    <STRING-TYPE> |
    <INTEGER-TYPE> |
    <FLOAT-TYPE> |
    <LOGICAL-TYPE> |
    <NUMBER-TYPE> |
    <TYPE-ID>
```

==

```
<REFER-TYPE> ::=
    refer [<CONSTITUENT-LIST>]


<REPEAT-CONTROL> ::=
    [<INCREMENT-CONTROL>] [<WHILE-CONTRL>] [<UNTIL-CONTROL>]


<REPEAT-STATEMENT> ::=
    repeat <REPEAT-CONTROL> ;
    [<STATEMENT>]...
    end ;


<RESULT-TYPE> ::=
    <QUALIFIED-SEMANTIC-TYPE> |
    <ENTITY-ID>


<RULE-BODY> ::=
    <ACTION-BODY>


<RULE-CALL> ::=
    <RULE-ID> <PARAMETER-LIST>


<RULE-DECL> ::=
    <RULE-HEADER>
    <RULE-BODY>


<RULE-HEADER> ::=
    rule <RULE-ID> ( [<FORMAL-PARAMETER-LIST>] ) ;


<SCHEMA-BODY> ::=
    [<SCHEMA-INFO>]...
    <BLOCK-DECL>
    end ;


<SCHEMA-DECL> ::=
    <SCHEMA-HEADER>
    [<SCHEMA-BODY>]


<SCHEMA-HEADER> ::=
    schema <SCHEMA-ID> ;
```

===

```
<SCHEMA-INFO-KEYWD> ::=
     title |
     author |
     version


<SCHEMA-INFO> ::=
     <SCHEMA-INFO-KEYWD> <STRING-LITERAL> ;


<SEMANTIC-TYPE> ::=
     <COMPLEX-SEMANTIC-TYPE> |
     <QUALIFIED-SEMANTIC-TYPE>


<SKIP-STATEMENT> ::=
     skip ;


<STATEMENT> ::=
     <REPEAT-STATEMENT> |
     <CASE-STATEMENT> |
     <IF-STATEMENT> |
     <COMPOUND-STATEMENT> |
     <WITH-STATEMENT> |
     <PROC-CALL-STATEMENT> |
     <ASSIGNMENT-STATEMENT> |
     <ESCAPE-STATEMENT> |
     <SKIP-STATEMENT> |
     <DUMMY-STATEMENT>


<STRING-OP> ::=
     <CONCAT-OP>


<STRING-TYPE> ::=
     string [<PRECISION-SPEC>] [varying ]


<STRUCTURE-TYPE> ::=
     structure ;
     <ATTRIBUTE-DECL-STMT>... ;
     end


<TYPE-SPEC> ::=
     <IDENTIFIER-LIST> : <QUALIFIED-SEMANTIC-TYPE>


<UNTIL-CONTROL> ::=
     until <CONDITION-SPEC>
```

==

<WHERE-CLAUSE> ::=
    where <RULE-CALL> [, <RULE-CALL>]...

<WHILE-CONTROL> ::=
    while <CONDITION-SPEC>

<WITH-STATEMENT> ::=
    with <IDENTIFIER>
    <COMPOUND-STATEMENT>

BASIC CONSTRUCTS

```
<ATTRIB-ID> ::=
    <IDENTIFIER>


<CHARACTER> ::=
    <LETTER> |
    <DIGIT> |
    <SPECIAL-CHAR>


<CLASS-ID> ::=
    <IDENTIFIER>


<EMBEDDED-REMARK> ::=
    (* [<CHARACTER>]... *)


<ENTITY-ID> ::=
    <IDENTIFIER>


<ENUM-ID-LIST> ::=
    ( <ENUM-ID> [, <ENUM-ID>]... )


<ENUM-ID> ::=
    <IDENTIFIER>


<FIXED-LITERAL> ::=
    <INTEGER-LITERAL> . [<INTEGER-LITERAL>]


<FLOAT-LITERAL> ::=
    <FIXED-LITERAL> e [+ | - ] <INTEGER-LITERAL>


<FUNC-ID> ::=
    <IDENTIFIER>


<ID-CHAR> ::=
    <LETTER> |
    <DIGIT> |
    _    >> The Underscore Character <<


<IDENTIFIER> ::=
    <LETTER> [<ID-CHAR>]...
```

```
<INTEGER-LITERAL> ::=
    <DIGIT>...


<LETTER> ::=
    <UPPER> |
    <LOWER>


<LEXICAL-ID> ::=
    # <IDENTIFIER>


<LOCAL-ID> ::=
    <IDENTIFIER>


<LOGICAL-LITERAL> ::=
    true |
    false


<OPER-ID> ::=
    <IDENTIFIER>


<PROC-ID> ::=
    <IDENTIFIER>


<REMARK> ::=
    <EMBEDDED-REMARK> |
    <TAIL-REMARK>


<RULE-ID> ::=
    <IDENTIFIER>


<SCHEMA-ID> ::=
    <IDENTIFIER>


<SIGN> ::=
    + | -


<STRING-LITERAL> ::=
    <QUOTE> <CHARACTER>... <QUOTE>

    >> If a quote is to be part of a string, then use two <<
    >> consecutive quotes. e.g., 'John''s book              <<
```

```
<TAIL-REMARK> ::=
    -- [<CHARACTER>] <NEWLINE>

    >> Two consecutive hyphens start a tail remark <<


<TYPE-ID> ::=
    <IDENTIFIER>


<UNSIGNED-NUMBER> ::=
    <INTEGER-LITERAL> |
    <FIXED-LITERAL> |
    <FLOAT-LITERAL>


<WHITE-SPACE> ::=
    <SPACE> |
    <TAB> |
    <NEWLINE> |
    <REMARK>
```

BASIC ELEMENTS

<DIGIT> ::=
      0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<LOWER> ::=
      a | b | c | d | e | f | g | h | i |
      j | k | l | m | n | o | p | q | r |
      s | t | u | v | w | x | y | z

<NEWLINE> ::=
      >> Implementation Dependent <<

<QUOTE> ::=
      ' >> Apostrophe <<

<SPACE> ::=
      >> Blank Space Character <<

<SPECIAL-CHAR> ::=
      >> Any Character That Is Not A <LETTER> Or A <DIGIT> <<

<TAB> ::=
      >> Implementation Dependent <<

<UPPER> ::=
      A | B | C | D | E | F | G | H | I |
      J | K | L | M | N | O | P | Q | R |
      S | T | U | V | W | X | Y | Z

OPERATOR EQUIVALENCE TABLE

OPERATOR	SYMBOL
<ADD-OP>	+
<SUB-OP>	-
<MULT-OP>	*
<REAL-DIV-OP>	/
<EXP-OP>	**
<IN-OP>	in
<INT-DIV-OP>	div
<MOD-OP>	mod
<CONCAT-OP>	||
<LESS-THAN>	<
<LIKE-OP>	like
<GREATER-THAN>	>
<EQUAL>	=
<RANGE>	range
<LESS-EQUAL>	<=
<GREATER-EQUAL>	>=
<NOT-EQUAL>	<>
<AND-OP>	and
<NOT-OP>	not
<OR-OP>	or

===

TABLE OF KEYWORDS

A	ARRAY	ASSOC	AUTHOR
B	BEGIN BY	BETWEEN	BOUND
C	CASE	CLASS	
D	DEFINE		
E	ELSE ENUMERATION	END ESCAPE	ENTITY
F	FIXED	FLOAT	FUNCTION
I	IF IS	INPUT	INTEGER
K	KEY	KEYROLE	
L	LIST	LOCAL	LOGICAL
N	NOROLE	NUMBER	
O	OF	OPERATION	OTHERWISE
P	PROCEDURE		
R	RANGE ROLE	REFER RULE	REPEAT
S	SCHEMA STRUCTURE	SKIP	STRING
T	THEN TYPE	TITLE	TO
U	UNARY	UNTIL	UPDATE
V	VALUES VERSION	VAR	VARYING
W	WHERE	WHILE	WITH

Table of Contents

1. Syntax

The syntax of DSL is defined formally in Part II of this document. This part is aimed at explaining how the constructs are put together to create a schema definition.

DSL can be characterized as block structured and free form. The free form characteristic means that one statement may span several physical lines, or that several statements may be written on a single physical line.

Blocking gives scope to the names used to identify things. These names are not known outside the scope of the block in which they are defined.

A block is a collection of statements. Each block has a particular kind of statement (called a block header) that begins it. All blocks are explicitly terminated by the END statement. Some examples of blocks are:

```
SCHEMA block1;
  ENTITY block2;
    ROLE
       a : REAL;
       b : INTEGER;
  END;
  ENTITY block3;
    ROLE
       a : INTEGER;
       b : REAL;
  END;
END;
```

In this example block1 contains block2 and block3. Since the schema block is always the outermost block, the scope of its name (block1) is global. The two entities (block2 and block3) are at the same level and their names must be different. A name inside one entity block may be the same as a name in the other entity block as in the example shown. The attributes a and b of block2 have no connection with the attributes a and b of block3.

A statement is a collection of tokens (words, literals, punctuation) ending with a semicolon character.

Tokens make up the basic building blocks of DSL. All tokens are character strings that observe certain formation rules, always based on the first character, e.g., a word always begins with a letter of the alphabet, and no other token begins in that manner.

2. BASIC LANGUAGE ELEMENTS

2.1 Character Set

A schema written in DSL consists of a string of characters, which consists of letters of the alphabet, digits, and special characters.

Letters
A to Z and a to z

Underscore
_   (used as letter except as first letter in a word)

Digits
0 to 9

Special Characters
+ - * / = < > ( ) . , : ; ' # | ? blank

Compound Symbols

Certain symbols are made up of two or more characters.

<> <= >= ** || -- (* *) AND DIV IN LIKE MOD OR RANGE

No distinction is made between upper and lower case letters except when they appear within a quoted string.

2.2 Reserved Words

Reserved words are part of the DSL environment and may not be redefined, i.e., used as identifiers. Reserved words are classified as keywords, standard identifiers, and compound symbols.

══

2.2.1 Keywords

ARRAY	ASSOC	AUTHOR
BEGIN	BETWEEN	BOUND
BY		
CASE	CLASS	
DEFINE		
ELSE	END	ENTITY
ENUMERATION	ESCAPE	
FIXED	FLOAT	FUNCTION
IF	INPUT	INTEGER
IS		
KEY	KEYROLE	
LIST	LOCAL	LOGICAL
NOROLE	NUMBER	
OF	OPERATION	OTHERWISE
PROCEDURE		
RANGE	REFER	REPEAT
ROLE		
RULE		
SCHEMA	SKIP	STRING
STRUCTURE		
THEN	TITLE	TO
TYPE		
UNTIL	UPDATE	
VALUE	VALUES	VAR
VARYING	VERSION	
WHERE	WHILE	WITH

2.2.2 Standard Identifiers

The following standard identifiers are part of the DSL environment and may not be redefined.

FALSE	TRUE

2.2.3 Compound Symbols

Certain operator symbols are formed from letters and look like words. However, they might just as well have been represented as combinations of special characters. The compound symbols are:

AND	DIV	IN
LIKE	MOD	OR
RANGE		

2.3 Program Lines

A physical line ends with a newline character (either an explicit character or combination of characters, or in the case of fixed length records, the last character in the record). A single word or literal must fit entirely in a physical line. The effect of this is that the limitations of the implementation (with respect to line length) determines the maximum length of

identifiers and other elements of the syntax. Obviously, the
minimum possible line length must be greater than or equal to the
length of the longest reserved word.

3. USER DEFINED LANGUAGE ELEMENTS

3.1 Identifiers

An identifier is a string of characters (a word) used to name
some element of the schema. The first character in the string
must be a letter, and the remaining characters, if any, must be
letters, digits, or the underscore character.

No distinction is made between upper and lower case letters.
e.g., POINT, Point, and point are all the same identifier.

An identifier must not be the same as one of the DSL reserved
words. A table of reserved words may be found at the end of
Part II of this document.

Some examples of valid and invalid identifiers are:

valid
point
line
CenterOfGravity
end_point_2

invalid
 _abc   -- starts with underscore
6end   -- starts with digit, not letter
continuity flag   -- has embedded space

3.2 Literals

Literals are self defining constants used to form
expressions. The types of literals permitted by DSL are:

            Integer   Float
            Fixed   String
            Logical

3.2.1 Integer Literal
is a string of digits. A unary sign[+-] operator may be used
to negate the number. The unary operator is not part of the
literal however.

═══

3.2.2 Fixed Literal

is an integer literal followed by a decimal point and an optional integer literal. The scale of the fixed number is taken to be the total number of digits in the number and the precision is taken to be the number of digits to the right of the decimal point. A unary sign[+-] operator may be used to negate the number. The unary operator is not part of the literal however.

Examples of valid and invalid fixed literals are:

valid
```
-123.45        scale=5    precision=2
0.01           scale=3    precision=2
4.599999       scale=7    precision=6
4.             scale=1    precision=0
```

invalid
```
-.45           No leading digit
4. 56          Embedded blank
4              Not fixed, treated as integer
```

3.2.3 Float Literal

is an integer literal followed immediately by a decimal point, an optional integer literal, the letter E, and an integer literal which is the exponent. No spaces are permitted within this string. A unary sign[+-] operator may be used to negate the number. The unary operator is not part of the literal however.

The first part (before the E) defines the mantissa of the number. The last part (after the E) defines the exponent.

Examples of valid and invalid float literals are:

valid
```
-1.E16         -10000000000000000.0
568.99E-3                    0.56899
0.001E5                    100.000
```

invalid
```
-1E16          No decimal point
568. 99E-3     Embedded blank
-.001E5        No leading digit
```

3.2.4 String Literal

is any string of characters enclosed by single quotes. A string literal is the only context where upper and lower case letters are significant. i.e., 'ABC' is not equal to 'abc'.

If a single quote is part of the literal it is written twice. e.g., 'John''s dog' is interpreted as John's dog.

The string literal may contain escape characters to represent special symbols, e.g., Kanji or mathematical symbols. At the present the escape mechanism is not defined.

3.2.5 Logical Literal

is either the word TRUE or FALSE.

3.3 Lexical Substitution

The author may define character strings to be substituted within the body of the schema. This is done by:

```
DEFINE
   #abc = any text you wish ;
```

The substitution string consists of the first significant character through the last significant character between the equal mark and the semicolon mark. Note that a semicolon may not be part of the substitution string. In this example the substitution string is: any text you wish.

```
ENTITY qwerty;
   #abc ;
```

Each time #abc is seen it is replaced by the text defined in the DEFINE block. Note that this is a simple text substitution and no checks are made (at the time of the substitution) for correctness. However, when the substituted text is processed, any errors will be flaged.

The example would yield:

```
ENTITY qwerty;
   any text you wish ;
```

which, of course, would produce an error when it is processed.

Substitution does not take place within string literals, or within comments. Thus,

```
       a = '#abc';
          or
       (* #abc *)
```

would not produce a substitution.

3.4 Comments

Comments may be used to document the schema. The entire comment, including the (*, *), or -- symbols are ignored. They may be written in two ways.

Embedded comments follow the form:

> (* anything you wish to say *)

(with the exception of *) which must not be part of the comment), and tail comments follow the form:

> -- anything you wish to say <newline>

Embedded comments may be written any place a blank character is permitted and is treated as a blank.

Tail comments must appear at the end of any significant part of a statement as the double hyphen and all of the remaining characters on the same physical line are ignored.

4.   SCHEMA

The schema is everything known about some enterprise of interest. It includes entities, classes, rules (and other algorithms), and operations. It is expressed in a way that is independent of possible implementations. It is structural, but not structural in the sense of a given implementation, i.e., the mapping of relational tables from the schema will yield an entirely different structure.

A schema is written as:

```
SCHEMA schema-name;
   -- type declarations
   -- entity declarations
   -- class declarations
   -- rule declarations
   -- etc.
END;
```

The elements of the schema must be defined before they are referenced, except that entities may make forward references to classes and vice versa.

Each element of the schema must have a name that is different than every other name known to the schema, e.g., if there is an entity named point there may not be a class named point.

5. ENTITY

An entity is an abstraction of something about which information is to be kept. Generally speaking an entity represents some artifact. An entity is made up of a set of attributes which are the facts about it. Each attribute has a name and a semantic type. Attributes may also have either static or dynamic constraints. These constraints define the set of values that are permissible for a given fact.

The names of attributes must be unique within the scope of the entity in which they are declared. Attribute names do not have to be unique within the schema. The name of an entity may be the same as the name of one of its attributes (since the entity is in the scope of the schema, while the attribute is in the scope of an entity). Thus:

```
ENTITY a;
   a : integer;
END;
```

is permissible (although perhaps poor style).

Attributes are classified either as role, norole, or associative (assoc).

A role attribute is an essential fact about the entity. Without it the meaning of the entity is impossible to determine.

A norole attribute exists only for the purpose of supporting some implementation. A norole attribute could be removed from the declaration without affecting the meaning of the entity.

An associative attribute defines a special relationship between two or more other entities. All associative attributes must be of the semantic type refer. Like role attributes, associative attributes are necessary for understanding although they are not strictly part of the definition of the entity.

An entity is written:

```
ENTITY entity-name [WHERE constraint];
NOROLE
   -- norole declarations
ROLE
   -- role declarations
ASSOC
   -- association declarations
END;
```

The different kinds of declarations may be given in any order.

An attribute is declared as:

attribute-name : semantic-type [WHERE constraint];

See Section 8 for a discussion of semantic types. The
constraint is optional. When given, a combination of static and
dynamic constraints may be specified. See Section 9 for static
constraints.

<u>examples:</u>

```
1    ENTITY point;
2    NOROLE
       name : STRING;
3    ROLE
       x,y,z : REAL;
4    END;

5    ENTITY line
       WHERE disjoint(defining_position_1,defining_position_2);
6    NOROLE
       name : STRING;
7    ROLE
       defining_position_1 : REFER(point);
       defining_position_2 : REFER(point);
8    END;
```

1) begin definition of entity called 'point'.
2) begin norole declarations, i.e., 'name' which is a string
   semantic type.
3) begin role declarations. This also ends the norole
   declarations. x,y,z are the names of the role attributes
   all of which are of the semantic type real.
4) ends the entity definition.
5) begin definition of entity called 'line'. The where clause
   calls for a dynamic rule (not shown) that says the two
   defining points of a line must be disjoint.
6) begin norole declarations, i.e., 'name' which is a string
   semantic type.
7) begin role declarations. This also ends the norole
   declarations. defining_position_1 and defining_position_2
   are both references to a point (entity). An alternate
   definition strategy is shown following. This definition
   says that the defining points may be shared (i.e., used in
   the definition) by other entities.
8) ends the entity definition.

Alternate Definition of Line

```
   ENTITY line
     WHERE disjoint(defining_position_1,defining_position_2);
   NOROLE
     name : STRING;
9  ROLE
     defining_position_1 : point;
     defining_position_2 : point;
   END;
```

9) here we define line by two points, not references to points. This says that the two defining points may not be used by other entities. This definition has a stronger sense of dependence than the case given next. The same disjoint rule would apply just as in the other case.

```
   ENTITY line
     WHERE disjoint(defining_position_1,defining_position_2);
   NOROLE
     name : STRING;
10 ROLE
     defining_position_1 : DEPENDENT REFER(point);
     defining_position_2 : DEPENDENT REFER(point);
   END;
```

10) here we define by references to point, but say the points may be shared (since there is an instance of each point, they may be referenced by anything). However, if we delete an instance of line, we must also attempt to delete both instances of point. This attempted delete may or may not be possible depending on how those two points are otherwise used.

6. CLASS

A class defines a grouping of things that are thought to be alike in some manner. When reference is made to a class it means that any member of that class will satisfy the reference. e.g., if we define the class:

```
   CLASS fruit;
     OF (apple, orange);
   END;
```

and then make reference to fruit, we are saying that either apple or orange is a permissible reference. A class is a good way to mix apples and oranges so to speak.

A class may have other classes as members. We could define:

```
CLASS tart_fruit;
  OF (lemon, grapefruit, cranberry);
END;
```

and

```
CLASS sweet_fruit;
  OF (pear, strawberry, apple);
END;
```

and then:

```
CLASS fruit;
  OF (tart_fruit, sweet_fruit);
END;
```

A class is defined by the block:

```
CLASS class-name;
  OF (member-list);
END;
```

class-name is a valid identifier which names the class being defined. The member-list is a list of names of classes or entities, that are members of the class being defined. Each member in the list is separated by a comma and each member must be of the same type (i.e., they all must be classes, or entities, etc.).

7. ALGORITHMS

DSL provides for four kinds of algorithm definitions. A procedure is a general algorithm which performs some sequence of operations. A function is a procedure that yields a particular result which may be either a simple type or a complex type. A rule is a special kind of function that yields only a true or false result. An operation is a way of expressing a result in infix or prefix style. It is, in fact, a way of extending the set of built-in operators of the language.

7.1 Procedure

A procedure is written in the same manner as ordinary procedural languages. The general form of the procedure block is:

```
PROCEDURE proc_name( formal-parameters ) ;
LOCAL
  any local declarations go here
BEGIN;
  body-of-procedure
END;
```

A procedure is invoked by the call:

```
proc_name( actual-parameters ) ;
```

The number and type of the actual parameters and formal parameters must match exactly. A procedure may be written without formal parameters, in which case the procedure header and invocation would look like:

```
PROCEDURE proc_name();
BEGIN;
END;
```

```
proc_name();
```

That is, the left and right parentheses are not optional.

7.2 Function

A function is a procedure that produces the specified result type. The result type may be simple or complex. The invocation of a function is exactly equivalent to writing a variable of the given type in an expression.

7.3 Rule

7.4 Operation

Built-in operations such as add, subtract, multiply, and divide are commonly written as:

[1]   a + b * c / d

but could also be written:

[2]   add(a,divide(multiply(b,c),d))

taking into account the usual hierarchy of operation.

Whichever style is used depends somewhat on individual preference and accepted practices. We (most of us) would prefer to see the arithmetic expression written as style [1] instead of style [2].

The operation construct allows for extension of the built-in set of operations such that the written language reflects the style that is most natural to the user community.

An operation is written:

OPERATION infix(id:type; symbol; id:type):type;
BEGIN;
END;

-or-

OPERATION unary(symbol; id:type):type;
BEGIN;
END;

where:

infix or unary are the names given to the operation being defined (actually names like add, invert, cross, age, etc. would be used. Infix and unary refer to the characteristic of the operation itself).

type is either the input or result objects, which may be simple objects such as numbers or strings, or complex objects such as entities. id is the local identifier of the object for reference within the algorithm.

symbol is a string literal that defines a new reserved word. Even though it is written as a string the following restrictions apply:

-- no embedded blanks are permitted

-- the first character must be a letter [A..Z, a..z]

-- the remaining characters must be letters, digits, and underscore

-- the symbol becomes part of part of the DSL environment and has global scope.

Once an operation has been defined it may be used in the same manner as built-in operations. First vector dot product and vector cross product operation will be defined, then they will be referenced to illustrate this construct.

```
ENTITY vector;
ROLE
  i,j,k : FLOAT;
END;

OPERATION vector_dot_product(a:vector,'dot',b:vector)
  : real;
LOCAL
  c : real;
BEGIN;
  c = a.i * b.i + a.j * b.j + a.k * b.k;
  vector_dot_product = c;
END;

OPERATION vector_cross_product(a:vector,'cross',b:vector)
  : vector;
LOCAL
  c : vector;
BEGIN;
  -- body of procedure
  vector_dot_product = c;
END;

PROCEDURE some_name;
LOCAL
  v1, v2, v3, v4 : vector;
  dotpr : real;
BEGIN;
  -
  -
  -
  dotpr = v2 dot v3;
  -- operations are processed left to right. thus:
  v1 = v2 cross v3 cross v4;
  -- is the same as:
  v1 = (v2 cross v3) cross v4;
  -- nesting may be used to alter the left to right order
  v1 = v2 cross (v3 cross v4);
  -
  -
  -
END;
```

7.5 Expressions

Expressions are constructs giving rules for the computation
of values. They consist of operands: variables, literals, and
function invocations, combined by operators as explained in the
following paragraphs.

Each operator has a priority which determines when a particular subexpression is evaluated.

1. Exponentiation Prefix-   Prefix+   Not
2. Multiplication Division
3. Addition   Subtraction
   String Concatenation
4. Relational
5. And
6. Or

Expressions are evaluated from left to right taking into account the priority of each operator. i.e., each subexpression of priority 1 is evaluated from left to right; then each subexpression of priority 2 is evaluated in the same manner; etc. until the entire expression is finished.

7.6 Built-in Functions

A number of conceptual functions are built into the DSL environment to operate on numbers, strings, etc.

Number Functions

ABS(n)
COS(n)
SIN(n)
TAN(n)
LOG(n)
LOG10(n)
SQRT(n)
SQUARE(n)

8. Semantic Types

Semantic types are to information modeling as data types are to data modeling. In data modeling we are concerned with the form and character of data. In information modeling we are concerned with the abstract representation of information.

To illustrate this difference consider: in information modeling the notion of integer is simply that of a whole number. There is no assumption about how that whole number might be represented. In data modeling we become concerned with the details of implementation. e.g., do we use twos complement binary representation using four bytes? do we use numeric characters? do we use packed decimal format? and so forth. As a practical matter the difference between data type and semantic type is probably not important.

DSL supports the following semantic types:

```
Number --------\
   Integer     |--Simple types
   Real        |
      Fixed    |
      Float    |
String         |
Logical -------/

Array ---------\
List           |--Grouping type
Set -----------/

Refer -----------Relational type

Defined ---------Extension type

Structure -----\--Complex types
Enumeration ---/
```

The simple types correspond to the data types found in almost
every programming language or database management system. The
grouping types provide for homogeneous grouping of other semantic
types (including perhaps grouping types). The relational type
provides for forming relationships between things. The extension
type provides for an extension of the set of semantic types.
Complex types provide for heterogeneous groupings or arbitrary
naming of values. The complex types must be treated as extensions
of the semantic type set, i.e., they may not be used directly by
attributes.

In all of the examples the attribute declaration

                  example : semantic-type;

will serve as an example to show the proper usage each semantic
type.

8.1 Simple Types

```
Number
   Integer
   Real
      Fixed
      Float
String
Logical
```

Number is an abstraction of integer or real. It is an inexact specification. It is used when the exact specification is unknown, i.e., it is not known yet whether an attribute is integer, fixed, or float.

<center>example : number;</center>

Integer defines an attribute to be a whole number either signed or unsigned. This is an exact specification. An optional precision may be specified along with the integer specification. If no precision is given the integer is assumed to have an indefinite number of (decimal) digits. If the precision is provided the value of the integer has an implied constraint on the values an instance of the attribute.

<center>example : integer;
example : integer(p);</center>

where p is an unsigned integer literal.

Real is an abstraction of either a fixed or a float number. It is an inexact specification which is used when it is not known whether the exact type is fixed or float.

<center>example : real;</center>

Fixed is a number that has an exact or implied precision and resolution. The precision is the number of decimal digits in the number and the resolution is the number of digits to the right of the decimal point. Fixed numbers would commonly be used for money transactions for instance. A fixed number may be specified:

<center>example : fixed;
example : fixed(,r);
example : fixed(p,r);</center>

where p,r are both unsigned integer literals.

Float is a number that has an exact or implied mantissa precision and an exponent value of arbitrary magnitude. It is commonly used for scientific calculations. A float number may be specified:

<center>example : float;
example : float(p);</center>

where p is an unsigned integer literal which defines the minimum number of decimal digits in the mantissa.

String is a sequence of characters, which are normally printable. A string would be used to represent a persons name, address, etc. A string may also hold special non-printable characters which are used to control a printer or to represent special symbols such as those used in mathematical formulas or those needed for non-English alphabets (e.g., Kanji).

> example : string;
> example : string(l);

where *l* is an unsigned integer literal which defines the maximum length in characters of a particular string. Note that *l* corresponds to the number of characters, not the number of units of storage needed to represent the string value. e.g., a single Kanji character may occupy two bytes of storage.

<u>Logical</u> represents the states True or False. This is often called a *Boolean* value. It is specified as:

> example : logical;

8.2 Grouping Types

> Array
> List
> Set

The grouping types provide for homogeneous groupings of semantic types.

<u>Array</u> is a homogeneous group that has a defined lower and upper bound, which must be integer values. The lower bound must be less than or equal to the upper bound. Given that *L* is the lower bound and *U* is the upper bound, then there are exactly

$$U - L \text{ elements in the array.}$$

<u>List</u> is an ordered homogeneous grouping that has a defined lower bound, but a possibly indefinite upper bound. The lower bound specifies the minimum number of elements that are required to be in the list. This lower bound must be an integer which is zero or greater.

<u>Set</u> is an unordered homogeneous grouping that has a defined lower bound, but a possibly indefinite upper bound. The lower bound specifies the minimum number of elements that are required to be in the set. This lower bound must be an integer which is zero or greater. A *set* is similar to a *list* with this exception:

The order of the elements in a list are significant. The order of the elements in a set are not significant. Thus, if we had:

```
ENTITY group;
ROLE
   member : set(0 to ?) of refer(geometry);
END;
```

we would assume no significance by the order or position of one element of the set. But if we had:

```
ENTITY loop;
ROLE
   member : list(2 to ?) of refer(edge);
END;
```

the position of one edge within the list is significant, i.e., the edges must be ordered.

9. STATIC CONSTRAINTS

A static constraint specifies the values that a simple semantic type can (is permitted to) attain. Static constraints may be ranges of values, discrete values, or value bounds. This does not preclude the use of rules when specifying static constraints. It is recommended that rules be used when there are a number of static constraints on a particular semantic type.

9.1 RANGE is used to constrain values to a discrete range. For example, an integer number is valid only if it has the values 1 thru 10. The specification allows for inclusive and/or exclusive ranges. The specification format is:

```
RANGE ( lo-value <   value <  hi-value )
RANGE ( lo-value <=  value <  hi-value )
RANGE ( lo-value <   value <= hi-value )
RANGE ( lo-value <=  value <= hi-value )
```

lo-value and hi-value must be literals and must correspond to the semantic type which preceeds this specification. The semantic type may be any simple semantic type except for logical. lo-value must be less than hi-value.

Note that value is a keyword which equates to the name of the attribute being constrained.

examples:

x : real range(0 < value < 100);

This reads: x is a real number which must satisfy the condition 0<x<100 (zero is less than x, and x is less than 100).

i : integer range(1 < value <= 10);

This reads: i is an integer number which must satisfy the
condition 1<i<=10 (1 is less than i, and i is less than or equal
to 10). Since integers have a discrete resolution this could also
be written as:

i : integer(2 <= value < 11);

with no loss of meaning. Note that this is only possible with
integer and fixed semantic types. Number, real, and float are
assumed to have an unknown resolution.

s : string(4) range('A' <= value <= 'zzzz');

The value of s must satisfy:

if 'A' <= s then
  if s <= 'zzzz' then
    result := true;

This assumes that if the two comparands are of different
lengths the shorter is padded on the right with blanks to the
length of the longer before the comparison is made. The result of
the comparison will depend on the character set being employed -
ASCII, EBCDIC, etc.

9.2 BOUND defines a value half space. e.g., only the values
greater than 10 are valid for a given semantic type. The
specification follows the form:

BOUND (value rel-op constant);

where rel-op must be one of:

    <   <=   >   >=

The relational operators = (equal) and <> (not equal) are not
permitted in this context. The constant must correspond to the
declared semantic type which preceeds this specification.

examples:

x : real bound(value > 0);

y : real bound(value >= 0);

s : string bound(value > ' ');

===

9.3 VALUES defines a set of discrete values. Each of the discrete values must correspond to the declared semantic type. The specification follows the form:

    VALUES (constant, constant, ...);

where each constant must correspond to the declared semantic type which preceeds this specification.

    examples:

    x : integer values(1,3,5,7,9);

    s : string values('JAN', 'FEB', 'MAR', ... , 'DEC');

9.4 Summary

    The specification of static constraints can help to define the context and use of an attribute declaration. These specifications do not suggest how the implementation might enforce the constraint, only that the constraint must be enforced somehow.

    More than one static constraint may be applied to a single attribute. e.g.,

    x : integer range(0<value<20)
              values(31,41,51)
              range(100<=value<=255);

    Note that static constraints can be used to determine the minimum storage required for certain semantic types. In the previous case it is obvious that the total range of values is 1 through 255, but with some gaps. This value can be held in a single *byte* of memory. But consider the following:

    y : integer(3) range(0<=value<=1000);

    This is a conflict because the precision specification is not compatible with the range; one says that y can hold the values -999 through 999 while the other says the values can range from 0 through 1000. Other conflicts or confusion can result from mixed specification. e.g.,

    z : real bound(value<0) range(10<=value<=100000);

    This is confusing. It either says z can have any value less than zero or any value between 10 and 100000. Or it can be interpreted as saying that z must be both less than zero and also between 10 and 100000 which is of course nonsense. To avoid this possible confusion in interpretation mixing of bound and range is not permitted. A rule could be written to handle that case.

    But what about a case such as the following.

==

    a : integer range(0<value<100) range(40<value<300);

    A particular value of a, say 35, would satisfy the first
specification but not the second. The strategy in this case is to
issue a warning and reduce the specification to a common range.

    a : integer range(0<value<300);

==

> "When I use a word," *Humpty Dumpty said in*
> *rather a scornful tone,* "it means just what I
> chose it to mean - neither more nor less."
>
> from "Through the Looking Glass"
> by Lewis Caroll

1-NF, 2-NF, etc.	The technique of organizing attributes into records such that maximum stability in the developed structures is achieved.
? (OPEN UPPER BOUND)	Used to specify a to-many relationship.
APPLICATION	A set of procedural or non-procedural code used to accomplish some end state.
ARRAY SEMANTIC TYPE	An ordered collection of some semantic type having a fixed lower and upper bound (index).
ATTRIBUTE	A fact about an entity.
BACK POINTER	A physical implementation that accounts for the users of a given entity. A way of accomplishing a many-to-many relationship.
BNF	Bachas-Nauer Form (or Bachas-Normal Form). A method of expressing the grammar of a language.
CARDINALITY	The number of entities involved in a given relationship.
CLASS	A collection of entities that are (thought to be) alike in some manner.
CONCEPTUAL SCHEMA	See ANSI/X3/SPARC DBSG
CONSTRAINTS	A condition of any kind which must be enforced on values of attributes.
COLLECTION SEMANTIC TYPE ...	A semantic type, similar to list, which requires that all elements are of the same kind.

===

DATA .... The representation of information
                             independent of its meaning. e.g., an
                             integer number.

DATA MODEL .... A method of expressing the data
                             content and relationships of a data
                             processing system.

DBMS .... DataBase Management System

DEFINED SEMANTIC TYPE .... A semantic type that is defined by
                             the schema author. An extension to
                             the built-in semantic types.

DEPENDENT .... A relationship attribute which
                             stipulates that the referenced
                             entity exist only for the purpose of
                             defining the refering entity.

DISCIPLINE .... A branch of knowledge or training.

DISCOVERY .... The identification of information
                             needed by an enterprise.

ENTERPRISE .... A business, or a significant segment
                             of a business.

ENTITY .... A thing of interest about which data
                             is to be kept.

ENUMERATION SEMANTIC TYPE .. An ordered collection of values, any
                             one of which being the value of an
                             attribute of that type at a given
                             instant.

ENVIRONMENT .... The total of circumstances
                             surrounding a system, specifically:
                             the combination of external or
                             extrinsic physical conditions that
                             affect the behavior of a system

EXCHANGE .... The process of moving information
                             from one environment to another.

EXTERNAL SCHEMA .... See ANSI/X3/SPARC DBSG.

FIXED SEMANTIC TYPE .... A real number which has a defined
                             precision (total number of digits)
                             and resolution (number of digits to
                             the right of the decimal point).

FLOAT SEMANTIC TYPE .... A real number which has a defined precision (total number of digits in its mantissa) and an exponent.

FUNCTION .... An algorithm which yields some result object.

INDEPENDENT .... A relationship attribute that stipulates that the referenced entity may exist without the existance of the refering entity.

INFORMATION .... Data plus the meaning connected with it. It is any kind of knowledge about things or concepts that is exchangeable among users. e.g., an integer number is the age, in years, of some person. database

INFORMATION MODEL .... What STEP calls the schema.

The process of formalizing the structure and semantics of data; i.e., information.

INSTANCE .... An individual entity in a physical database.

INTEGER SEMANTIC TYPE .... A whole number.

INTERNAL SCHEMA .... See ANSI/X3/SPARC DBSG.

KEY .... An attribute that is used to identify an instance of an entity.

LIST SEMANTIC TYPE .... An ordered collection of a given semantic type which may contain zero, one, or more elements. Similar to ARRAY, but may not have a negative lower bound and the upper bound may be indefinite.

LOGICAL SEMANTIC TYPE .... A semantic type which has only the values True and False.

MANDITORY .... A relationship attribute that stipulates that a value is manditory (not optional).

MANY-to-MANY .... A kind of cardinality that allows an indefinite number of entities on either side of the relationship. Not usually permitted by data models; not permitted by this information modeling technique. See back pointer.

MAPPING FUNCTION .... A function that transforms one schema to another schema, or transforms the instances of entities defined by one schema to instances defined by another schema. A translator.

NOROLE .... An attribute that is not needed for the meaning of an entity.

NUMBER SEMANTIC TYPE .... A semantic type representing any number format: integer, real, fixed, or float.

ONE-to-MANY .... A cardinality where one entity is related to zero, one, or many other entities.

ONE-to-ONE .... A cardinality where one entity is related to exactly one other entity.

OPERATION .... An action that may be applied to an entity which yields a known result.

OPERATION SET .... A set of operations that apply to a specific entity or class.

OPTIONAL .... A relationship attribute that stipulates that an attribute value may be non-existant. This does not imply that the value is unknown.

PROCEDURE .... An algorithm which operates on information to produce some desired end state.

PROVIDER .... The person or system that provides information to (for) the receiver.

REAL SEMANTIC TYPE .... A semantic type of which fixed and float are sub-types.

RECEIVER .... The person or system that receives information from the provider.

==

REFER SEMANTIC TYPE .... A semantic type used to establish a relationship.

REFERENCE .... An instance of a relationship between two entities. The mechanism for establishing this link is implementation dependent.

RELATIONAL TABLE .... One kind of data model in which data is arranged into columns and rows.

RELATIONSHIP .... A mechanism used to connect (link) two entities in some fashion.

ROLE .... An attribute that is necessary for the meaning of an entity to be complete.

RULE .... A constraint which must be enforced by the information processing system.

SCHEMA .... A description, or definition, of the data elements of interest to an enterprise. The schema may be cast as conceptual, external, or internal.

It is what we call the the information model. It embodies what we (think we) know about enterprise information.

SEMANTIC TYPE .... An abstraction of the physical representation of an attribute. i.e., the way we think of the realization of data.

SET SEMANTIC TYPE .... A semantic type, similar to list, in which the order of elements is not meaningful.

STRING SEMANTIC TYPE .... A semantic type representing an ordered list of characters, i.e., letters, digits, special characters.

STRUCTURE SEMANTIC TYPE .... A semantic type which is a set of attributes that together have a meaning.

SYSTEM .... A computer system.

==

S T E P ....

TRANSLATOR .... A mapping function. The application
                            that accomplishes translation from
                            one schema to another.

Appendix E

Paper On PDES Delivered at Federal Computer Conference

THE

PRODUCT DATA EXCHANGE STANDARD

(PDES)

BY

J. C. KELLY

SANDIA NATIONAL LABORATORIES

CHAIRMAN, PDES LOGICAL LAYER INITIATION TASK

PDES SCOPE AND OBJECTIVES

PDES stands for Product Data Exchange Standard. A long-term project currently exists within the IGES Committee to develop PDES. This project has two primary objectives:

1. to develop an exchange standard for product data in support of industrial automation

2. to represent the US position in the International Standards Organization (ISO) arena relative to the development of a single worldwide standard for the exchange of product data

A new standard is being developed out of the belief that no existing standard can be extended to support industrial automation sufficiently well.

"Product Data" is taken to be more general than "product definition data". It includes data relevant to the entire life cycle of a product; manufacturing, quality assurance, testing, support, etc. Development of an exchange standard for product data involves settling on a set of logical structures to contain the product data information, and also settling on the manner in which these structures will be implemented in computer form.

The ISO Technical Committee TC 184 (Industrial Automation Systems) and its Subcommittee SC4 (External Representation Of Product Model Data) are relatively new committees within ISO, SC4 having first met in July, 1984. There is agreement within the Subcommittee that a single worldwide standard for the exchange of product data is needed. The goal of the standard is "...the capture of information comprising a computerized product model in a neutral form...throughout the life cycle of the product". The name of the standard is to be STEP - Standard for the Transfer and Exchange of Product Model Data.

The PDES Project has been designated to take the leadership role in the development of STEP. It is the US intention that PDES and STEP will be identical. Participating countries to this date, besides the US, include: France, Germany, Japan, Switzerland, and the United Kingdom.

THE PDES RELATIONSHIP TO IGES AND ITS DATA EXCHANGE HERITAGE

The IGES Steering Committee has outlined the relationship that is to exist between the IGES and the PDES specifications. Work within IGES Technical Committees will simultaneously be directed toward future, upward compatible versions of IGES and also toward a draft version of PDES. This version of PDES is to be, at a minimum, functionally equivalent to the then-current version of IGES. PDES need not be directly upwardly compatible with IGES, but must accommodate a conversion path. An IGES Long Range Plan, now in draft, will further explain this relationship and other topics as well, such as test methodology and committee structure.

Other data exchange efforts besides IGES will affect PDES in one way or another. In broad terms, the legacy of some of these other efforts is as follows: The IGES efforts form one "tier", or logical grouping, of

efforts. These have data exchange between dissimilar interactive graphics CAD/CAM systems as their driving force. Early versions were implicitly targeted toward systems of the 70's and early 80's, and toward mechanical applications. Thus, from the start, 2D drawings, 3D wireframe models, and certain generative type surfaces were emphasized. With the exception of those entities expressly intended for the drawing application area (eg., linear dimension, angular dimension, general note, etc.), most entities were generic (eg., line, arc, composite curve, associativity), so that the intent of the exchanged data had to be imposed from outside the data itself. Typically, this would involve a human viewing a representation of the data on a graphics system, keying off such things as geometric shape or placement relative to the part itself. Thus, the early versions of IGES were intended for human oriented interpretation of the data rather than for automated interpretation and use of the data.

A second tier of efforts consists of the CAM-I XBF-2 effort, the IGES ESP effort, the ICAM PDDI effort, and the follow-on GMAP effort sponsored by the Air Force CIM Program. (These acronyms denote, respectively, Computer-Aided Manufacturing-International, Inc.; Experimental Boundary File-2; Initial Graphics Exchange Specification; Experimental Solids Proposal; Integrated Computer Aided Manufacturing; Product Definition Data Interface; Geometric Modelling Applications Interface Program; Computer Integrated Manufacturing.)

These efforts between them bring two innovations to the fore, and in effect usher in a more modern product data exchange era. The first innovation is the emphasis on a more complete definition of the shape of a part - that is, the emphasis on solid modelling, in which the set of spatial points occupied by an object is completely determined. In addition to this complete "quantitative" description of an object, some systems also provide a "qualitative" description by decomposing the object into topological entities such as faces, edges, and vertices which describe the connectivity of the part. These two types of descriptions are then related by having the topological entities indicate which geometry entities are needed for their definition.

The PDDI project went a bit further than just geometry and topology entities, identifying entities for other higher level qualitative structures called part or form "features". Features allow high-level concept communication about parts. Examples are hole, flange, thread, web, pocket, chamfer, etc. The PDDI feature entities relate specific topology and geometry entities to a given feature so that identifying information for that feature can be explicit in the data, a necessary condition for the support of automation. Geometry, topology, and feature information are often collectively referred to as "shape information".

The other innovation from this class of data exchange efforts, in fact, from the PDDI and the GMAP efforts, is the emphasis on having the computerized part model be a "complete" model. This means that the model contains all necessary shape and non-shape information sufficient to accomplish a given function; that the information is in a suitable, i.e., automation-enabling, computer form; and, that the different types of information are associated as required. For example, tolerance information would be carried in a form directly interpretable by a computer rather than in a computerized text form intended primarily for interpretation by a

human, and, this information would be associated with those entities in the model affected by the tolerance. Other non-shape entities include administrative entities having to do with such things as effectivity, specifications, material, notes, etc. Thus, the general notion associated with a complete part model is that it obviates the use of human-oriented drawings and other paper documents as a necessary means of passing information between different functions.

It is interesting to note that the first tier of efforts are all standards efforts, concerning themselves with existing systems and techniques, while the second tier of efforts are research and development projects concerning themselves with finding out how things should be, and ultimately intending to effect change.

THE CONTENT EMPHASIS OF PDES

The PDES effort will reflect this dual heritage, and will extend it. It is intended that the PDES effort will also have a proactive influence on both users and vendors.

PDES will extend the heritage from the standards efforts and the research and development efforts by providing a means for an organization to communicate its product breakdown structure. This implies accommodating such notions as part, subassembly, assembly, version, effectivity, release, etc., and also accommodating the natural correspondence between these kinds of items and the configuration documents, test data, change directives, etc., that pertain to these items. Many questions remain to be answered here. For example, a way must be found to relate the product breakdown structure to the PDES file or files representing that product (as when a model has to be spread over several files, or several models are contained in one file), and to do this in a way that will serve diverse companies that have diverse needs in this area.

THE PDES METHODOLOGY AND ITS CHALLENGES

The distinguishing characteristics of the PDES methodology reflect recent developments in data base and information systems in general. They also reflect techniques used and experience gained in other data exchange efforts. The PDES methodology is significantly different from the IGES methodology.

The PDES methodology involves: a three-layer architecture similar to the three-schema framework for data base management systems as identified by ANSI/X3/SPARC; reference models; formal languages; and coordination with other standards efforts.

Three-layer Architecture

The three-layer architecture is similar to the three-schema framework in which external, conceptual, and internal schemas are identified. In that framework, the conceptual schema comprises the unique central description, from the standpoint of the enterprise, of the meaning of the data, and the relationships among, and constraints upon, the data. It embodies the

"rules of the business". This description is computer independent, i.e., it is conceptual. The external schema represents the manner in which individual users and applications need to view the data represented by the conceptual schema. Each external schema can therefore be supported by the one conceptual schema. The internal schema represents the actual physical computer storage structure being used to store and access the data. Within PDES, the three layers corresponding to these three schemata are the logical layer, the application layer, and the physical layer.

The application layer will contain the descriptions and combinations of information peculiar to various application areas. Information modelling techniques (or, data modelling techniques, as they may sometimes be called) will be used to formally express what the required pieces of information and their relationships are for a given application area. These models are examples of what are termed "reference models".

This layer will be supported by application subgroups such as the standing subcommittees now in IGES: Advanced Geometry, Electrical, Mechanical, AEC, FEM, Drafting, etc. The challenge here will be to actually do the modelling, then to manage the networking of the models into "clusters" depending on the product under consideration. Consistency and sufficiency within each cluster must be insured. The modelling itself will be a challenge because the application of information modelling techniques to production artifacts seems to be a new area.

The purpose of the logical layer is to provide a consistent, computer-independent description of the data constructs that will contain the information to be exchanged. Both generic and application-specific constructs are expected to be identified. The central challenge here, and perhaps in the entire PDES effort, will be to devise and carry out a conceptualization and integration methodology by which a minimally redundant set of generic data structures and relationships can be produced. That is, this set must be as lean as possible, and at the same time sufficient to support the wide range of applications. Some experience will be needed to be able to settle on such a methodology, but it will likely be a combination of a bottom-up approach (i.e., abstracting from information about individual application areas) and a top-down approach (i.e., deducing needed structures and relationships starting from some global classification schema for product data). Another challenge will be to build into the methodology the flexibility of being able to consistently extend the schema to accommodate new applications. Establishing modelling technique requirements is also expected to be challenging.

The physical layer corresponds to the internal schema and will be concerned with the data structures and data formats for the exchange file itself. The main challenge here will be to establish and maintain efficiency in such areas as file size and processing time.

Reference Models

A reference model for some universe of discourse is a model that collects together the necessary pieces of information and also their relationships to each other. The notion includes some mechanism, usually graphical, for describing the pieces of information and the relationships.

Reference models will be used throughout the PDES architecture. The purpose is to promote thoroughness in domain analysis and precision in definition and communication of information, especially between different layers.

The first challenge here for a standards group that historically has been concerned with product data exchange issues is simply to learn something about reference models and information modelling, and about the requirements that any particular technique must satisfy in order to be useful. Another challenge is to effectively communicate the substance of these issues to people who are familiar with information modelling, but are probably not familiar with product data exchange, and to do this in a way that results in new talent being brought to bear on our problems.

Formal Languages

Formal languages will be used for the definition of data structures and for the PDES file syntax. Emphasis will be on languages with context-free grammars so that parsers can be built more simply.

Coordination With Other Standards

A final characteristic of the PDES methodology will be its relationship with other standards efforts, both national and international. How data is represented within PDES, as well as what data is represented will be coordinated with other efforts to insure compatability and to minimize duplication.

Experience from the IGES efforts has shown that segmentation of the development of a standard along the lines of the three-level architecture would be a desirable thing. In the IGES efforts, there was no explicit segmentation. One result was that, in the course of pursuing new application areas, application-oriented people were being held responsible for things outside their area of expertise, such as the formulation of physical data structures. Another result was that there was no one explicitly charged with maintaining a global viewpoint toward the entity set and toward the consistency of the makeup of new entities. Thus, for example, relationships between application-specific entities and generic entities may not be consistent whenever these are used together. In some cases the application specific entity may be subordinate, while in other cases the generic entity may be subordinate. Along the same lines, it was difficult to insure that the generic entity set was kept as lean and minimally redundant as possible.

PDES VERSION 1.0 - PROPOSED CONTENTS

The proposed contents of PDES Version 1.0 as of this time are as follows:

Application Layer:

1. Mechanical Products - Several classes of parts modeled, including the classes of machined, turned, flat, and sheet

2. Electrical/Electronic Products - Electrical Schematics, Printed Wiring Board Physical Design

3. Architecture, Engineering, and Construction (AEC) - Heating, Ventilating, Air Conditioning (HVAC) Distribution Model

4. Finite Element Modelling (FEM) - Nodes, Elements, Loads/Constraints, Displacements, Postprocessing

5. Drafting

Constituent Technical Areas

1. Manufacturing Technology - Administrative Data, Tolerances

2. Solid Modelling - Boundary Representation (B-Rep.), Constructive Solid Geometry (CSG)

3. Curve And Surface Modelling - Wireframe Geometry, Surface Geometry

4. Presentation Data - Viewing pipeline, View Mechanism, Text Definition

Logical Layer - Develop conceptualization and integration methodology, and apply to application reference models

Physical Layer - Develop ASCII file format for a single PDES file

PDES PROOF OF CONCEPT WORK NOW UNDER WAY

A PDES proof-of-concept and general learning exercise has been under way since January of this year. This work is collectively known as the PDES Initiation effort. The effort will involve all three layers of the architecture, but is being administered by two task groups. One is associated with the logical layer, and the other is associated with the physical layer.

The goal of the Initiation work for the physical layer task group is the specification of a file structure for the PDES exchange form. The current draft of the specification specifies the file structure as being language based, described by an unambiguous, context free grammar expressed in Backus-Naur form. The specification draws heavily on previous PDDI experience in this area.

The Initiation work for the logical layer is divided into two tasks. The

goals of the first task are to illustrate that a conceptual schema can be developed in support of a specific application area, and then to communicate this schema to the physical layer using a Data Specification Language (DSL). The master reference model for the conceptual schema will use the Nijssen Information Analysis Model (NIAM) information modelling technique, with the DSL description being generated manually from this.

The goals of the second task are to illustrate that the initial conceptual schema can be sequent ally augmented in a consistent manner to support additional application .reas, and then to communicate the augmented schema to the physical layer g oup.

The spirit of the initiation work is that we will do the best job possible and then will examine and evaluate the products of our efforts and our methodologies. For PDES longer term efforts, what is good will be retained and what is not will be discarded.

Many fundamental PDES topics are yet to be widely discussed. (Examples are requirements on pre and post processors for effective implementation of PDES, requirements on user databases to allow full advantage of PDES, and interplay between these.) The precise articulation and subsequent discussion of these topics should be an important part of the longer term PDES effort.

THE PDES SPECIFICATION - SUMMARY

1. PDES is being developed to support industrial automation. It will deal with the entire range of product data and will represent the US position internationally in the quest for a single standard.

2. PDES content will emphasize solid modelling, complete product models, and product breakdown structure. It is intended that PDES will have a proactive influence on both users and vendors.

3. The PDES methodology is significantly different from the IGES methodology, and offers many challenges. It is based upon a three-layer architecture, reference models, conceptualization and integration, and formal languages.

4. Proof of concept work is currently under way and is known as the PDES Initiation effort. Content and methodology beneficial to long range PDES work will be kept.

Appendix F

Logical Layer Charter

Charter

Of The

PDES Logical Layer Initiation Task Group

Submitted By:

J. C. Kelly
Sandia National Laboratories
Chairman, Logical Layer Initiation
Task Group

July 30, 1985

Charter Of The PDES Logical Layer Initiation Task Group

The PDES Logical Layer Initiation Task Group is an ad hoc subcommittee of the PDES Committee. This Task Group will perform work as described below as part of the PDES Initiation effort.

The purposes of the work of this Task Group are:

1. Examine the possibility and feasibility of developing PDES according to the three level ANSI/X3/SPARC architecture as suggested in the second PDES report.

2. Establish logical layer content which potentially could serve as a baseline for future PDES development.

The two main tasks of this Task Group are:

1. Illustrate that a conceptual schema can be developed in support of a specific application area, and communicate the structure of this schema to the PDES Physical File Structures and Formal Languages Task Group.

2. Illustrate that this conceptual schema can be augmented in a non-redundant manner on an application-by-application basis.

The first task will contribute to illustrating the use of all three levels of the three level architecture. The second task will illustrate that the conceptual schema can be incrementally augmented as the need arises - a characteristic of the PDES environment.

Each task will result in a communication of the conceptual schema content to the Physical File Task Group. A Data Specification Language will be used for this. Communication for the first task will be approximately mid-September, 1985, and communication for the second task will be approximately November 1, 1985.

Requests will be made of application groups to compose reference models to be used in the second task. These groups could conceivably be based in existing IGES Subcommittees, or could be ad hoc.

A final report will be written. The report will describe and document what work was performed, and will make recommendations based on this experience. The general time frame for this report is January 1, 1986.

The Task Group will meet as required in order to accomplish its work, and will periodically report on its progress.

Task Overview

Task 1 A conceptual schema will be developed to support the mechanical design of flat plates with circular holes. Wireframe geometry will be used. The schema will support some user-view presentation (viewing) scenarios pertinent to this area of mechanical design.

The wireframe geometry entities and the presentation entities will be developed as part of this task. A reference model describing the conceptual schema will be produced. The Information Analysis (IA) information modeling methodology will be used for this reference mode., and the Data Specification Language (DSL) description of the conceptual schema will be based on this reference model.

Task 2 Four Application Task Groups will be contacted to compose reference models. These models will be used one at a time to cumulatively augment the conceptual schema produced in the first task. A reference model depicting the "final" conceptual schema will be produced, as will a mapping illustrating how each application makes use of the conceptual schema. The cumulative augmentation of the conceptual schema will involve integration in the sense that a minimum number of "generic" entities and structures will be sought to support the common needs of the various applications.

The integration work is the focal point of this task. The Information Analysis (IA) modeling methodology, and associated Software Tools (ST), will be used to support this work. (Specifically, IAST, a CDC software product in the possession of those who will be doing the integration work, will be used.) In order to provide a common footing for the integration work, and to make possible the use of the supporting software, each reference model from an Application Task Group will be translated into an equivalent IA-based reference model, and entered into IAST. The resulting translated reference model will be scrutinized from a "Quality Assurance" point of view. A liason from the referring Application Task Group will assist in understanding and possibly refining the reference model, thereby closing the QA loop.

IA will be used to describe the final conceptual schema, and also to illustrate how each application makes use of the conceptual schema. As in the first task, the Data Specification Language description of the conceptual schema will be based on the IA reference model of the conceptual schema.

The Application Reference Models Are:

1. Mechanical Design - Flat Plates With Circular Holes

2. Electrical Design - Schematics

3. Tolerancing - Tolerances In Y14.5M And ISO 1101 and 1660

4. Finite Element - Finite Element Environment

5. AEC - AEC/HVAC

Recommendations For General PDES Evaluations, recommendations, and experiences based on this work will include:

1. An evaluation of the three level architecture as an environment for developing PDES.

2. Recommendations for a logical layer integration methodology.

3. Recommendations for application layer information modeling techniques.

4. Experiences with the use of automated tools.

PDES Logical Layer Initiation Task Group

I. Bodnar, CDC
D. Briggs, Boeing
R. Brown, Hughes (New Member)
E. Clapp, IBM, Wireframe Geometry Task Leader
S. DePauw, Caterpillar, Flat Plate Design Task Leader
R. Gale, DACC':
D. Hemmelgarn, ITI
J. C. Kelly, Sandia, Chairman
P. Kennicott, GE
H. Ladd, DuPont (New Member)
D. Schenck, McDonnell-Douglas, DSL Task Leader
D. Theilen, Allied/Bendix, Logical Layer Integration Task Leader
D. Winfrey, DEC, Presentation Task Leader
J. Zimmerman, Allied/Bendix

Application Layer Tasks Initiated By Request Of The Logical Layer
Initiation Task Group And Their Coordinators With The Logical Layer

1. Mechanical Design - Reference Model For Flat Plates
        S. dePauw - Caterpillar, Task Leader
        D. Hemmelgarn - ITI

2. Electrical Design - Reference Model For Schematics
        C. Parks - General Dynamics, Task Leader
        P. Kennicott - General Electric
        Work Supported By: IGES Electrical Subcommittee

3. Tolerancing - Reference Model For Tolerances In Y14.5M
        R. Colsher - IGES Data Analysis, Task Leader
        B. Burkett - McDonnell-Douglas
        Work Supported By: IGES Drafting Information Model WG

4. Finite Element - Reference Model For FE Environment
        R. Ivey - Westinghouse, Task Leader
        B. Freeman - Allied/Bendix
        Work Supported By: IGES FEM Subcommittee

5. AEC - Reference Model For AEC/HVAC
        F. Stahl, IBM, Task Leader
        P. Rourke, Newport News Shipbuilding
        J. Turner, University Of Michigan
        Work Supported By: IGES AEC Subcommittee

Appendix G

Response to Call For Alternative Modeling Languages

Date: March 5, 1985

To: Roger W. Gale, Chairman, PDES Reference Model Selection

From: J. J. Zimmerman, PDES Physical Layer Committee Member

Subject: SUMMARY OF NIAM MODELING METHOD

I understand the critical nature of selecting an adequate reference model for the PDES logical layer. In response to your call for alternatives to $IDEF_1$, I present this NIAM overview.

NIAM (Nijssen's Information Analysis Method) is a binary data modeling method developed in Europe by Dr. G. M. Nijssen in 1972. The method as it is used today is nearly the same as the 1975 version with some dialectical changes. It has been applied to application projects in Europe since 1975 and is well recognized and respected there. It has been used in the states, however, only since about 1980 and now several major U.S. corporations are using NIAM. Since 1984, Qint Database Systems Corporation has supported NIAM with QINT/IAM and QINT/TINA support tools. Since 1980, Control Data Corporation has supported NIAM with IAS (Information Analysis Support) which consists of education, consulting, project management services, and support software.

The method itself, its nomenclature, and graphical language are public domain and are well described by ISO working group ISO/TC97/SC5/WG3 in report SC5-N695. I understand this working group is now ISO/TC97/SC21/WG5-3 and the original report has been republished as report SC21-N197.

This working group was formed to bring some formality to the ANSI/X3/SPARC three schema DBMS architecture--particularly the conceptual schema. As you will notice in the report, the working group used NIAM as a representative for a broad category of models called binary models. The other categories were EAR (Entity Attribute Relationship--which I think is the category $IDEF_1$ belongs to) and IPL (Interpreted Predicate Logic). The fact that ISO chose NIAM as a representative for binary modeling indicates European recognition. I am sending you a copy of ISO report SC5-N695. You will find a pretty thorough comparison of the EAR and binary approaches which I

think represents a fair theoretical comparison of $IDEF_1$ and NIAM.

The following are the major strengths of NIAM:

o   It has international acceptance.

o   The method is in public domain.

o . There is a body of people with experience with the methodology in Europe and North America.

o   End users find it easy to learn.

o   The method has a rigorous basis in classical linguistic and mathematical theory. This rigorous basis is the foundation for a formal language description for NIAM. This formal language is computer sensible.

o   NIAM clearly separates the process of problem space semantic analysis from the design of logical and physical record structures.

o   Computerized tools are commercially available to support model creation, model access, and model conversion to logical record structures and physical file formats.

Both NiAM and $IDEF_1$ methods are striving for the same goal-- the rationalization of information in preparation for integrated system development. More specifically, in Appendix A I have summarized areas in which NIAM is superior to $IDEF_1$ as you requested in your letter.

In general, I believe NIAM is superior to $IDEF_1$ as a conceptual data modeling reference language for PDES logical layer development.

March 5, 1985

Page 3

We are prepared to bring more information to you in greater
detail. Thank you for you attention.

Address correspondence to:

John J. Zimmerman, MH39
Allied Bendix Aerospace
P. O. Box 1159
Kansas City, MO   64141

JJZ:mca/7

Attachment

Features provided by NIAM that are not provided by $IDEF_1$.

1) NIAM is an internationally recognized and respected modeling method. It is used by major corporations in North America and Western Europe. Its formal basis in linguistics and set theory are well recognized.

Why the difference is important:

PDES logical schemas must be communicated internationally. International recognition of the modeling language will be a critical factor in ISO negotiations.

2) NIAM is a binary entity-relationship modeling technique that clearly separates the process of determining meaning of entities and their relationships from the process of designing logical data records (the process of determing key structures, normalization, and inter-record linkages).

Why the difference is important:

The information analyst should be dedicated to the task of understanding the meaning and relationships of the entities. The thought process should not be split between semantic analysis and logical record design. Increased concentration on the meaning of the problem space is the most significant benefit to logical data modeling.

3) IAS software provides for the automatic conversion of the NIAM binary model representation to normalized logical record structures.

Why the difference is important:

Reduces human labor and human error.

4) IAS software provides for the automatic conversion of the NIAM binary model representation to target physical data representations. These converters are called "PIPES." Several DBMs pipes are already in existence.

Why the difference is important:

A. This capability allows the conceptual schema to stand alone, independent of any physical data representation. It allows for multiple experimental or optional physical file formats to exist. I believe there is significant potential to pipe NIAM constructs from the NIAM conceptual schema to the PDES physical transfer file format

1

without any other intervening formal conceptual languages.

B. The minimization of the number of conceptual languages used in building the PDES logical model should enhance efficiency of communication and reduce learning overhead for logical modelers.

5) The NIAM graphical language contains a rich set of inter-entity constraint constructs.

Why difference is important:

Explicit constraint representation in the model reduces the amount of narrative associated with the conceptual schema and provides improved communications by using a standard constraint notation. The end result is a conceptual schema that is richer in meaning without introducing interpretation error.

6) NIAM separates functional dependency notation from existence dependency notation. This modularity allows the analyst to use all 16 combinations of functional dependency (4) and existence dependency notation (4).

Why difference is important:

It allows the analyst to be more expressive by using any combination of functional and existence dependency. The analyst does not need to add extensions to the modeling language.

7) The NIAM model uses a simple set of rigorously defined model constructs that have foundations in classical linguistics. The four most significant constructs are LOT (lexical object type--classes of object representations), NOLOT (nonlexical object type--classes of objects), IDEA (relationship between two nonlexical objects) and BRIDGE (relationship between a LOT and a NOLOT).

Why difference is important:

A. These constructs form the basis for a simple but rigorous modeling language that is computer sensible. A computer sensible language provides capability for computer assisted access to complex model structures.

B. These constructs improve the analyst's ability to understand and denote the difference between an object and the object's name.

2

C.   Gives analyst the ability to rigorously denote synonyms, abbreviations, homonyms, and arbitrary lexical mappings.

8)   IAS software provides the automated capability to integrate complex functional area model views by resolving different functional area or problem specific naming conventions with synonym tables.

     Why the difference is important:

     Resolving naming differences within various PDES application reference models is a mandatory step in the process of model integration at the conceptual layer. Synonym capability also allows the conceptual modeler to use abstract naming conventions that will promote integration. This capability will allow separate groups to work independently and yet have their model intersections reconciled.

9)   IAS software provides a set of automated cross reference tools that serve functions similar to Entity Class/Attribute Class Matrix, Class Migration Index, Attribute Class Migration Class Index, and Relation Matrix.

     Why the difference is important:

     Building these kinds of cross reference tables manually as in IDEF$_1$ is extremely labor intensive and error prone. Analyst resistance to model changes will grow as manually derived tables become more complex and interweaved. The analyst should not be required to manually build model indices that can be built automatically.

10) NIAM allows for, but does not demand, refinement of relationships in which existence independence exists on both ends.

     Why the difference is important:

     It is understandable that the most stable information structure is desirable in the end result. It is unreasonable for the modeling language to dictate that all relationships be refined. The demand for mandatory refinement can often be arbitrary and place an unnecessary burden on the analyst. The modeling of physical artifacts in PDES will involve naturally occurring many-to-many relationships which have legitimate existence. Refinement is artificial except when an relationship is better thought of as an object (in the case when the relationship itself is associated with many entities).

Appendix H

Summary of Responses to Call for Alternative Modeling Languages

TO: PDES Committees and Working Groups

FROM: Roger W. Gale -- Logical Layer Reference Model
            Selection

SUBJECT: Comparison of Proposed Modeling Methods and
            Recommendations for Selection

Dan Appleton is fond of saying, "the methods used have more to do
with the outcome than the objectives". My experience leads me to
agree with him. I believe that we should be concerned not only
with the elegance of a selected model for the logical layer but
with the method used to arrive at the model as well.
Accordingly, I am biased toward how we are to arrive at the model
somewhat more than toward a resulting model which has one or two
more bells or whistles.

There are only two models which have been given support as
candidates for the PDES Logical layer reference model. The ICAM
IDEF1 model received four recommendations and the ISO TC97/SC5 -
N 695 received one.

There are differences in the methods behind the two models which,
I believe, affect their utility for the purposes of PDES. The
most significant difference is that, in essence, IDEF1 is a
"top-down" method while the ISO is a "bottom-up" method.

The advantages I see in the IDEF1 method are that it is
"top-down" and is more "people friendly". The "people friendly"
aspect is important.   The ICAM experience seems to have
demonstrated that the IDEF1 method assists people to arrive at a
conceptual model of the enterprise data. That model is of the
conceptual data view of the business we are in. It is not a
model of computer systems, but rather, a definition of
requirements for data integrity which must be met when computer
systems are implemented. It is not easy for people who have been
trained in classic, two-schema, computer systems development
methods to learn to think in terms of the third schema, the
conceptual enterprise model, rather than the bits, bytes,
records, fields, etc. of computer implementation. We need all
the help we can get because, in the end, it is the people who
must do it.

In addition, the PDES logical layer model is ultimately a part of
a larger enterprise model on which work already is underway in
many firms. Much of that enterprise modeling is already in the
IDEF1 method (for example, the ICAM, MFG1 model).

1

The disadvantage of IDEF1 may lie in the lack of a standard, structured definition language for text description of the model and those needed data constraints not shown in the graphics. This disadvantage may tend to go away if one of the software tools for IDEF1 models is used because some of those tools, such as the one from the Dan Appleton Co. use a structured modeling language to define the model which has similarities to the definition language found in the ISO Standard.

I have received from J. C. Kelly, "A Preliminary PDES Logical Layer Document" dated March 13 1985. Attachment G of that document is by Douglas Schenk. In it is a proposed specification language for the PDES Logical Layer which would describe a different model from the IDEF1 model. The concept of a structured model description language is well represented in his paper.

The methodology behind the ISO model is available with automation from Control Data. I think it can be summarized as an approach in which "attributes" are related to each other and the data structure is determined from the binary relationships. The model provides a graphic representation which is supported by a very specific, structured model language in which data constraints are defined.

An advantage of the ISO model is that the very structured model definition language is very attractive to computer system implementors. A disadvantage is that the graphics convey only a limited portion of the model and the language statements must be studied to find most of the data constraints. My personal reaction is that it is difficult to grasp the model because it is hard to see the data context in the business sense which should be a primary objective of the model.

The methodology behind the IDEF1 model is one of; identifying objects of interest (entities) about which the enterprise maintains data, determining how the enterprise relates those entities, identifying the attributes of the entities and then refining the model according to some simple rules to arrive at a normalized, relational model.

Software is available from several sources providing more or less assistance with developing the IDEF1 models. AUTOIDEF is basically a computer aided drafting tool for models developed under ICAM. Other software may be purchased from its developers. Some of the software provides extensions beyond the basic IDEF1 which produce graphic illustration of additional data constraints.

Action is underway now to produce a new version of IDEF1 which will contain additional model conventions to illustrate data constraints. One effect of the new version will be to reduce the difference between the two methods in terms of the degree of data constraint definition.

According to Dave Theilen, John Zimmerman with him at Bendix, is of the opinion that he will not find it difficult to translate from one model form to the other in either direction. John has considerable experience with the Control Data modeling software and some experience with the IDEF1 model. It is Dave Theilen's intent to perform those translations in the course of integrating the various PDES application models into the logical layer. If that proves successful, we may be able to take advantage of desirable features of both methods.

Based upon my understanding of the differences and utility of the two methodologies, and the expressions of support from the IGES community, I recommend the following:

1. Adopt the IDEF1 modeling methodology for the PDES logical layer (utilizing new versions as they become available).

2. Pursue the possiblity of translating the IDEF1 model into an ISO model and, if found feasible, make that a practice for PDES.

3. Ask Douglas Schenk to revise his specification language so that it is a description of an IDEF1 model plus those additional data constraints considered necessary but not contained in the IDEF1.

4. Develop written additions to the documentation of the IDEF1 methodology to describe the methods to be used to arrive at the definition of those additional data constraints found necessary for item 3. above.

It appears to me that there is some confusion regarding the roles and contents of each of the three schemas identified by ANSI SPARC which are the three layers proposed for PDES. It is going to be difficult to go forward with a three-schema idea if everyone has a different idea if what is in each schema. There is a need for people working on a schema to understand what it is and what it is not.

It should be understood that the IDEF1 method is for modeling the Logical Layer (conceptual schema), or portions of it. There appears to be some confusion about the use of this modeling method to model applications. Applications are in the realm of the ANSI SPARC external schema or user views. IDEF1 is for modeling data. External schemas contain information as well as data. Information is derived from data through some sort of algorithm. For example, a persons age is information derived from current date and the persons date of birth. The IDEF1 model can contain "current date" and "persons birth date" but not "age".

The task of each application sub committee should be to examine their application and produce an IDEF1 model which identifies that set of entities, attributes and relationships of the logical layer (Conceptual Schema) which are necessary for the application. This might be more easily understood as a sub-schema of the logical layer.

I have not found much work relating to methods for modeling, or describing, a "user view" or external schema which are, I believe, other names for an application.

The choice of description of the physical layer (or internal schema) is dictated by the choice of implementation. The description of a CODASYL data base differs from that of a relational database which is different from that of a hierarchical database.

Roger W. Gale
Roger W. Gale

Appendix I

Letters on Critical Issues

Written During Initiation Effort

PRODUCT DEFINITION DATA SCOPE

The PD in PDES stands for Product Definition. It is easy to speak freely of PDES but it is difficult to establish just what is Product Definition. In this paper I will offer a few thoughts about what sorts of data might be within the scope of Product Definition.

For the moment, let us consider four classes of data. The first I will name Product Data. This is data about the objects to be manufactured. Product Data is mostly setting requirements stated in terms of functional and physical characteristics which should be present in the objects when they have been manufactured. It includes text and geometry data as well as alpha-numeric data. The second I will call Production Data. Closely related to Product Data, this data describes how the objects are to be manufactured. The third I will call Operational Data. This data is closely related to Production Data but describes the events of production, such as lot size, schedule, sequence of assembly, etc. The fourth class I will call Resource Data. This data is closely related to Operational Data, but it describes the resources that are involved in operations, e.g., machines, people and money.

Product Definition Data probably includes all Product Data, most Production Data, some Operational Data and little or no Resource Data.

Now let us think about a method for documenting the Product Definition Data Scope and establishing its content.

For years we have considered the use of computers from the standpoint of automating particular processes. Planning methodologies were based upon establishing the relationship of data to processes. There is already an activity going on related to STEP wherein a very complicate matrix is being developed essentially relating processes to data modified by life cycle stages.

Now that we have data management technologies such as data base managment systems, we can realize that data is a shareable asset which can be considered on its own merits. This is as true of the data which describes product as it is of any other data within an enterprise. I suggest that we should consider the Product Definition scope as a data scope. In the Data Scope we become interested in how data are related to data.

The tool for describing a data scope is the same tool that we can use to define the Logical Layer of PDES. That is a logical data model. The data model which, with its companion method, has already been most widely used for CIM data in this country is IDEF1. There is an extended version of the model now in work which provides improved definition of data rules and constraints. However, when we are constructing a Data Planning Model we do not need to identify all of the attributes of entities and fully normalize the model. It should be enough to resolve non-specific relationships and establish the Key attributes of the Entities. We call this a Key-Based model. The examples I use will be in the D. Appleton Co. Data Modeling Technique which is essentially the same as the extended IDEF1.
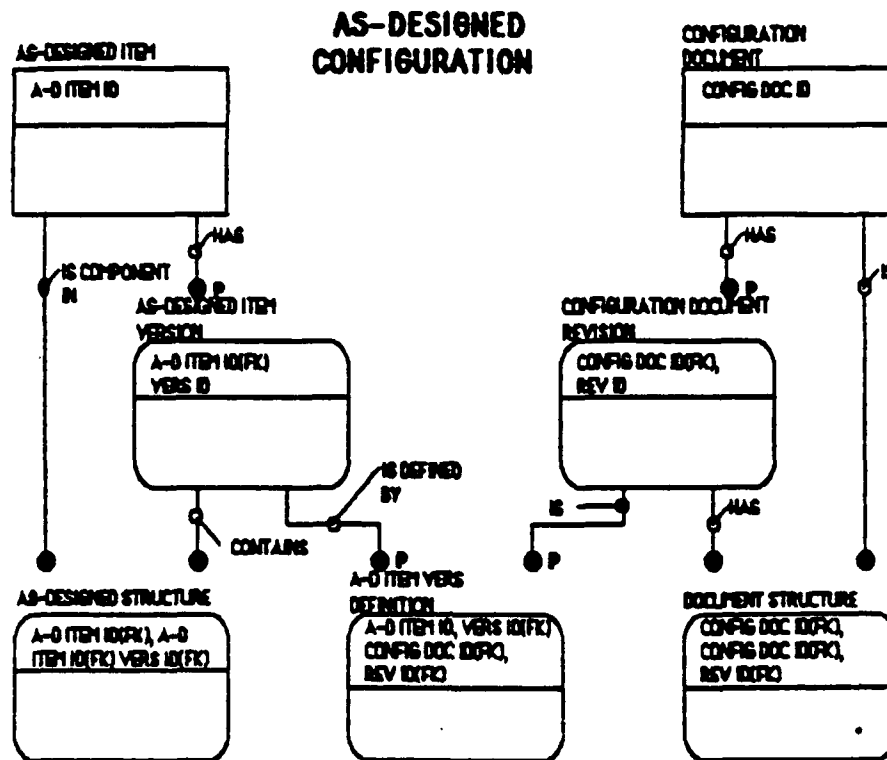
PRODUCT DEFINITION DATA SCOPE

If we begin with the design of a product we find that, until the advent of computers, the product "as-designed" exists as a set of documents which completely describe the functional and physical characteristics of the product and each of its subdivisions. For the subdivisions we have used a lot of terms such as assembly, subassembly, part and material. I will use the single term "item" to include all of these.

Let us begin with some simple definitions and business rules:

1. An Item may contain other Items as components.

2. An Item may be a component of other Items.

3. A document which contains all or part of the characteristics of an Item is a Configuration Document.

4. A Configuration Document may invoke other Configuration Documents to establish Item characteristics.

5. A Configuration Document may be invoked by other Configuration Documents.

6. Every Configuration Document has at least one Revision, the original, and may have subsequent Revisions.

7. Different Revisions of a Configuration Document may invoke different Configuration Documents.

8. Any Revision to a Configuration Document for an Item results in a Version of the Item.

9. Every Item has at least one Version, the original, and may have subsequent Versions.

10. Different Versions of an Item may have different Components.

11. A Configuration Document defines characteristics for one or more Items.

From these rules we can construct the data model of Figure 1. I believe that this is the most pivotal portion of the Product Definition data model. All of the early efforts to establish the requirements which the design must satisfy drive toward the "as-designed" product. In turn, it drives the subsequent manufacturing actions which first establish the "as-planned" configuration and finally result in an "as-built" configuration.

PRODUCT DEFINITION DATA SCOPE

FIGURE 1

I believe that a primary hurdle confronting CIM is to successfully capture the "as-designed" product definition and transfer it in a manner that allows automated generation of the "as-planned" configuration.

We have started to capture some of the Product Definition in CAD systems and IGES was an early effort to aid automated transfer of data from the CAD systems. However, the data in today's CAD systems is far from a complete Product Definition. Furthermore, the data which are present usually require a human in the loop for interpretation. For example, tolerances have usually been expressed in such a way that we cannot make a computer program interpret them to automatically control the output of an NC machine.

Let me suggest here that Configuration Documents may come in a variety of media which can include digital data. I think that a CAD system file may be one of those documents in the data model of Figure 1.

So far, I believe that the focus of IGES and of PDES has been on the very difficult task of understanding and defining the data by which we describe the shape of Items. In Figure 1 we have a data model which will provide a context into which we may fit the shape-defining data models on which we have concentrated.

Another key element of Product Definition is the directive that specific versions of "as-designed" Items should appear in specific units of delivered product. This is generally known as Effectivity. Effectivity usually is assigned at the time of Release. Release, in concept, seems to be establishing a status for a configuration of an Item which permits, or directs, manufacture of the Item.

PRODUCT DEFINITION DATA SCOPE

There seems to be a popular opinion that what we release is Documents. A good friend of mine in the Configuration Management business suggests that what we really ought to be releasing is Item Versions. In Figure 2, I have modeled the release of Item Configuration Documents which seems to be an attempt to do both. However, this model would only release Configuration Documents to the degree that they define specific Items. Items defined on the Configuration Documents which have no product usage are not released.
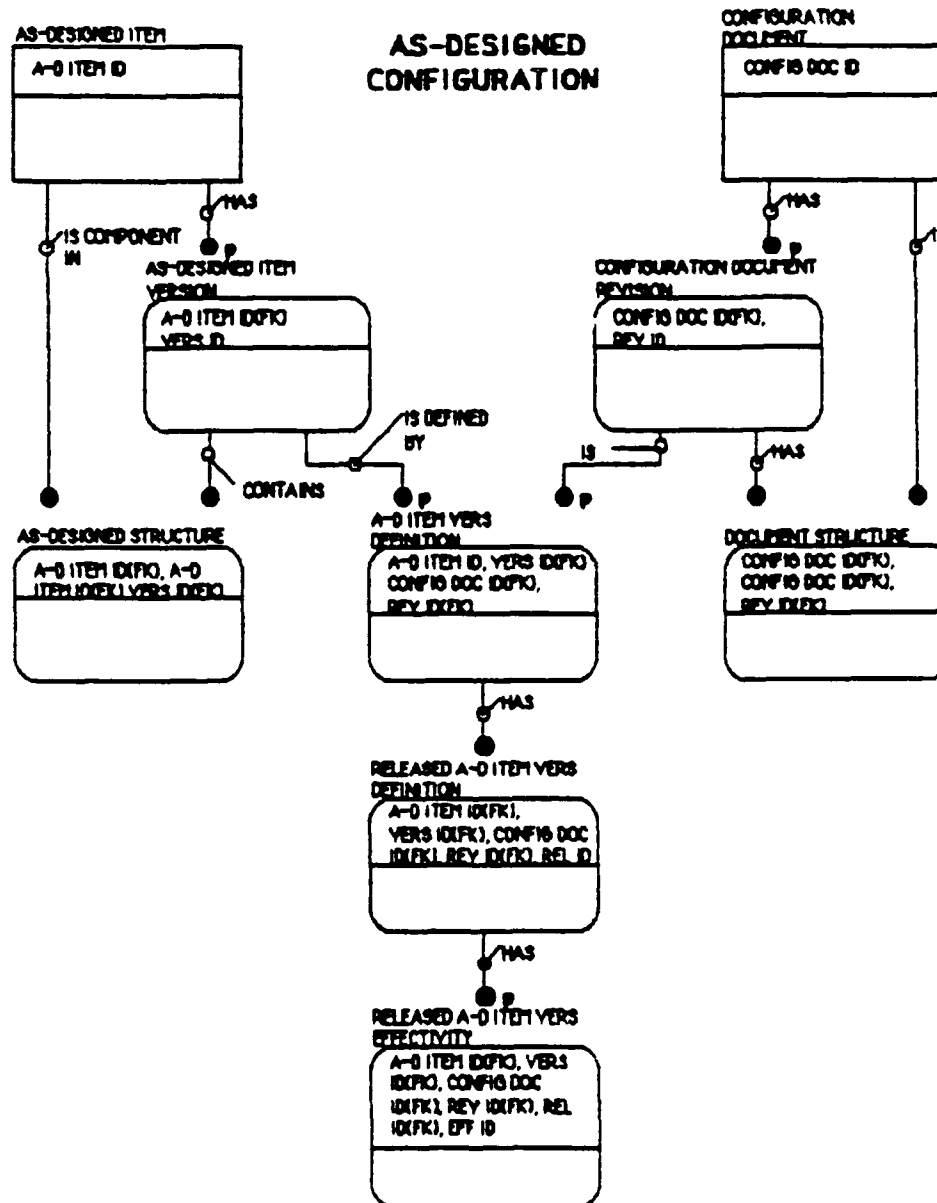
FIGURE2

PRODUCT DEFINITION DATA SCOPE

Now let me take this Product Definition data model a step or two further. It is common for the people planning the manufacture of something which engineering has designed to find that they cannot manufacture items in exactly the manner in which they are described in the design. For example, the designer of a multi-layer printed circuit board treats the board as a single item for which he describes the circuit paths to appear on each of the layers. The manufacturing folks have to take the multiple layers apart and make several items, each with layers of circuit on each side, and then bond them together to arrive at the item the design engineer described. Thus, manufacturing has several items where the designer had one. These "phantom" or "synthetic" items appear in the "as-planned" product structure for manufacturing and cause it to contain items not identified as items in the "as-designed" product structure.

The result of the planning process is that there is an "as-planned" set of Items and Configuration Documents which has a data model structure like that of the "as-designed" but is, fact a different structure with some differing content. The Configuration Documents for these Items are documents created in the manufacturing planning process; operation and routing sheets is one term for such documents. These documents usually will refer to the "as-designed" Configuration Documents to establish the configuration chain. The Entities and Attributes of this structure must be given unique names and definitions. The logical relationship of the "as-designed" and "as-planned" structures needs to be established. Figure 3 suggests a possible logical relationship.

During the design process, we typically evaluate the functionality of the Items by modeling or simulating the performance of the Item based on the characteristics established in the Configuration Documents. This is the design process equivalent of the performance of tests on actual product units once fabricated. The evaluations and their results should be related to the specific versions of the Items to which they apply and should be considered a part of Product Definition Data.

After Release, formal procedures and documentation are required to alter Configuration Documents (at least for the "as-designed"). There usually is a sequence of a Request for Change which, if approved, results in some sort of Change Directive. The Change Directive results in specific revisions of one or more Configuration Documents and represents those revisions until such time as the directed changes are incorporated into the contents of the Configuration Documents. These Change Directives are part of Product Definition.

During the course of manufacturing, Items come into being which do not conform to the characterictics within the limits prescribed by the Configuration Documents. Some of those non-conforming Items are found acceptable for use "as-is". The "as-built" configuration documentation seems then to consist of the "as-designed" plus differences introduced in "as-planned" plus the documentation of Items having acceptable variances from the prescribed limits.

PRODUCT DEFINITION DATA SCOPE
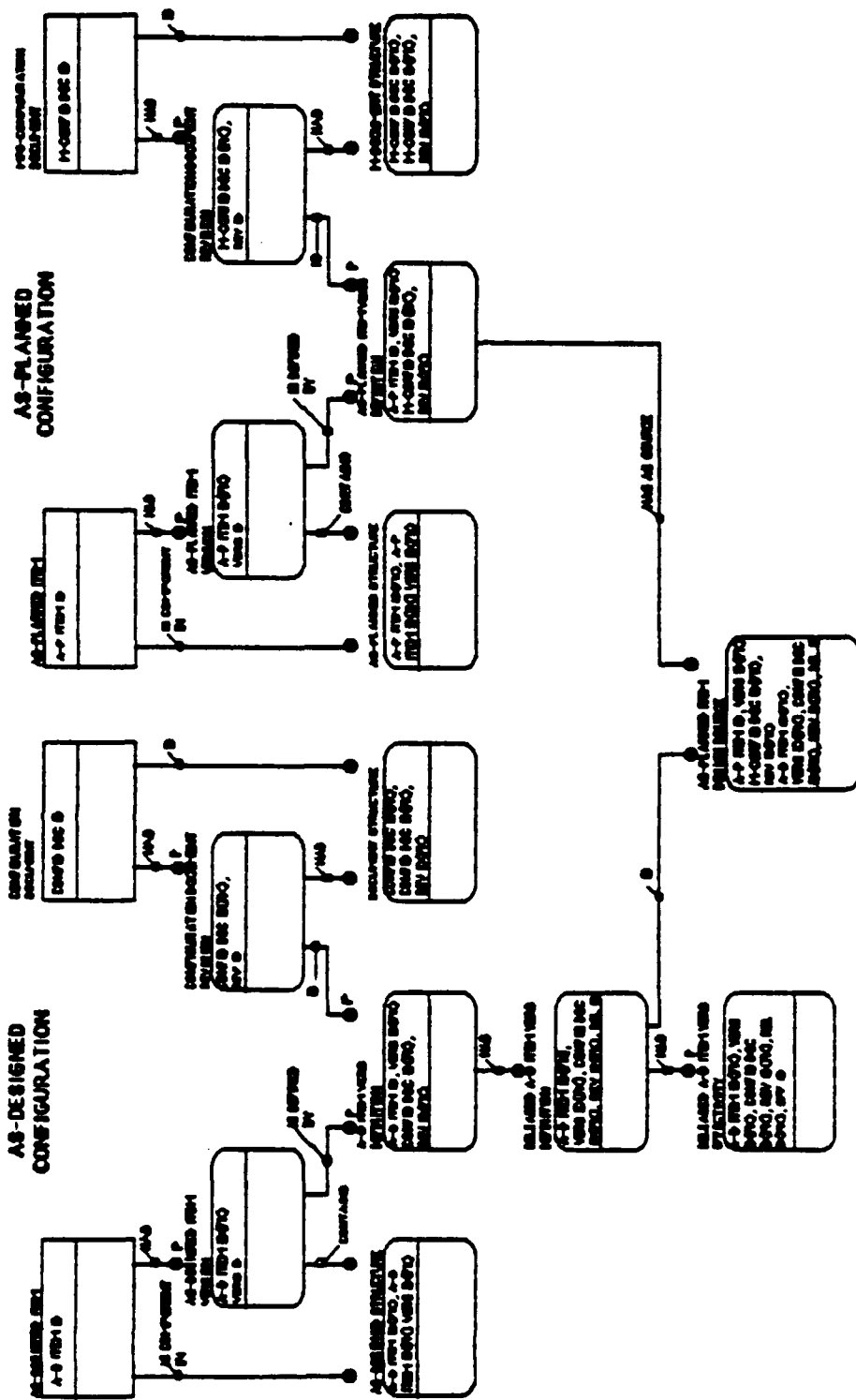
FIGURE 5

PRODUCT DEFINITION DATA SCOPE

With Figure 3 we now have a model of a central structure for Product Definition Data. Other Product Definition Data probably has direct or indirect relationships to entities in this central structure.

The "as-designed" and "as-planned" Items are actually "logical Items". That is, the Item Identification represents the set of characteristics which a Physical Item should have whenever it actually is fabricated. When we move to the data about "as-built" we are now trying to keep track of what is known about Physical Items. This is not hard if each Physical Item has a unique identifier which distinguishes it from its siblings such as a serial number. We do not serialize most Items. Instead we fabricate them in bunches, or manufacturing lots, and it is the group that we keep data about and what we know is mostly statistics about groups of items. It will take a little work and considerable thought to produce a logical data model in this area which will satisfy the rules for such models.

I have constructed a model which may provide the core data relationships upon which we can construct the relationships of the other Product Definition Data. What other data?? Well, here is where you have to start thinking.

I will finish by pointing out that the models I have included are what we call Data Planning models. They have only been developed to indicate basic relationships and entities. They are key-based. That is, the entities have been tested for validity by establishing that they do have keys and the keys have been migrated through relationships as required by the methodology. However, these models are still quite abstract. Giving attributes to the entities and applying the rules to attributes will cause the models to grow additional entities, perhaps by a factor of three to five.

Anyone wishing to discuss these ideas may call me at the phone number below or write to the listed address and I will try to respond.

Roger W. Gale
Consultant
D. Appleton Company, Inc.
1104 Highland Ave., Suite I
Manhattan Beach, CA 90266
(213) 318-2451

THE DOCUMENT PROBLEM
IN
PRODUCT DATA MODELING

There are two major forces at work driving most efforts to model Product Definition Data. One of these is the "Paperless System"; the idea that we should be able to eliminate the need for paper copies and use electronic copies of data instead. Many of the people talking about 'paperless" environments have not given a great deal of thought to what that might mean. Many of them in their minds see images of familiar documents on a CRT or other electronic display device. The more advanced thinkers see only that data needed to answer a particular question rather than the complete contents of one of today's documents.

The second force is Computer Integrated Manufacturing (CIM). Here the visions are of automated systems translating designs into manufactured articles with very little or no human intervention. To accomplish this end, we will either have to give the automated system the perception, intuition and reasoning powers that people have, or alter the way the data is presented and combined to define the product so that it can be interpreted correctly by computers.

When we are working on the Product Definition Data Model we keep bumping against the issue of "documents". In one sense documents are external views of data. In another sense, many of the documents are conceptual entities.

As an example, take "Drawing". Drawing seems to pass the tests as a conceptual entity because it is a thing, it is a thing that the enterprise keeps data about (schedules, completions, releases, revisions, etc.) and each instance has a unique identifier. A drawing is an aggregator of requirements (characteristics) for items. The manufacturing enterprise has developed significant business systems to utilize the drawing as the means of managing some of the physical and functional characteristics of its products. Other documents, such as specifications, are used to manage other characteristics.

When a team starts modeling product definition data, they soon reach "drawing". When we try to factor the drawing into its data elements, we find a variety of data and information. There are such elements as a drawing number and title. There are general notes stating requirements in a narrative form. There are one or more "views" of the shape definition. There may be various kinds of annotation associated with elements of the shape definition. There may be a bill of materials listing items which are constituents of the item being depicted by the drawing.

The IGES specification provides for a drawing, drawing views, flag notes and general notes as entities and that thinking carries over into the PDES initiation thinking.

The PDES Logical Layer committee has found the "Drawing" and is developing data models related to presentation of "views". There have been discussions as to whether or not the "presentation" is a legitimate part of a conceptual schema or is it not a user view or external schema in the sense of the ANSI/X3/SPARC definitions.

It is probably the case that there are both a conceptual drawing which is an aggregation entity for data which defines item characteristics and a drawing presentation which is a user view of the drawing data. We have a hard time accepting these as different and separating one from the other.

In order to achieve an understanding of a TO-BE, conceptual drawing, we need to set some sort of discriminator in place in our minds. Currently, I find it useful to ask myself if the conceptual data model has a simple physical implementation which could be computer interpreted. This leads to realization that "drawings" of the future are likely to be different from those of today in more ways than just being in digital format.

For example, "everbody" knows that drawings have general notes. I can model the statement that a drawing has zero, one or many general notes. That is small use to the developer of automated planning systems when what a general note says is, "faces marked P shall be cadmium plated per 00-P-XXX, type II, class 3 and shall be painted with paint conforming to MIL-P-NNNN, color number 276, except that crosshatched areas shall be plated but not painted".

With the present state of the art we are going to have to have a person interpret that note if we are going to plan the production of the part.

In order to make "drawing" interpretable by computer, we are probably going to have to make our electronic drawing so that associations such as those between surfaces and their required finishes are clearly understandable to the automated planning system. This is going to require some cultural change in the enterprise. In the Defense industry, the culture, standards and specifications of the customer will also have to change.

The "drawing" is only one sample of the "document" problem. How should we attack that problem?

I think that we are unlikely to be able to take the "giant step" to the future where the business is managing the product data as electronic data and paper documents have become merely one of the ways to present the data. That is particularly true when we consider that we are evolving from paper documents to electronic data. Some of each will continue to exist for some time to come.

In the beginning we will probably have to treat the paper documents as business entities. For such entities, our conceptual model will be of the data the business relates to the documents. This data is much like the data found in a library card catalog about the library collection. There is the data required to select the document appropriate to the need and data about the location where the document can be found. As we apply automation to the generation of Product Definition, we will begin by treating the resulting data collections as electronic documents and maintain and model the data in much the same manner as for the paper documents.

Gradually, as we begin to better understand what it is that we are doing, we may commence to model and manage the product definition data in a way that capitalizes upon the available computer technology. This will require changing the thinking of many people which is not a trivial task.

The PDES project is grappling with the Conceptual Schema of some of the data which is traditionally part of the drawing content. The prime focus is on the definition of an item shape. This is data which has been captured in Computer Aided Design systems. There shape is constructed from geometric elements such as lines, arcs, splines, etc. PDES is building upon the PDDI work which concluded that in order for computers to readily interpret the geometric shape data, it is necessary to collect it into topological entities such as edges, vertices, faces and shells.

Another look toward the TO BE suggests that the Group Technology ideas of Form Features such as holes, pockets, slots, grooves, etc. may also be used to collect geometric elements and give them meaning.

The original versions of IGES were based upon the desire to find an intermediate format for the transfer of data presently found in CAD/CAM systems. I believe that the primary motivation of PDDI was to find a more efficient way to do what IGES did and add some ability to transfer data not neccessarily found in existing CAD systems.

Neither of these projects has demonstrated much thinking or modeling of the relationships of the data they have modeled to other product definition data.

When we examine the PDDI and PDES thinking, we still find that they tend to model documents and include presentation elements in their conceptual schemata. For example, they provide for "color" attributes which have no bearing upon the meaning of the data. The "views" found in a drawing tend to appear as proposed conceptual entities. If there is a three-dimensional model of the shape of an item, views are not required to define the item. Views are then a projection of the three-dimensional shape onto a two-dimensional surface for presentation to people.

Another problem encountered when we are trying to model documents rather than data is that a particular kind of document may have come to be used for many different purposes in an enterprise. For example, in one enterprise, an Engineering Order (EO) is a document. It is used to record the release of up to ten drawings. It is also used as the formal release of changes to the content of a drawing prior to incorporation of the changes in a new version of the drawing. It is used as a "stop order" to halt fabrication or procurement of parts. Occasionally the EO is even used as a request to purchase parts.

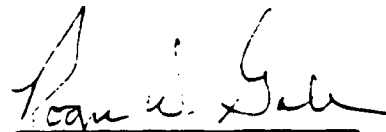**DACOM**
D. Appleton Company, Inc

When we try to model something like this EO we usually decide that these different things documented on an EO are actually different categories of the EO. When we are working on a conceptual schema without the handcuffs of documents we are more likely to model an "engineering release", a "manufacturing stop", a "procurement stop", a "change order" and a "purchase order request" as separate and distinct business entities.

DACOM
D. Appleton Company, Inc

October 22, 1985

I had a number of conversations in Knoxville representing some disquiet about the Steering Committee policy statements about the relationship between PDES and IGES. As a result I wrote the attached memo.

I am sending an info copy to everyone I can remember having such conversations with so that you can see my thinking written down.

I would appreciate your views on what I have said.

Roger W Gale
The D. Appleton Co., Inc
1104 Highland Ave.
Manhattan Beach, CA 90266

Phone: 213-318-2451

To: IGES Chairman and the PDES Project Manager

From: Roger W Gale

Subject: Steering Committee Policy Statement that PDES Version 1.0 Shall Have the Same
         Functionality as the Then Current IGES Version

The Steering Committee has issued a policy statement that version 1.0 of PDES shall have the
same functionality as the then current version of IGES. The problem is the definition of
"functionality". If it is taken to mean, "the ability to transfer application data found in
computer graphics files", then I contend that we do not know what the "functionality" of an IGES
file is because there is no knowledge of the meanings of the entities contained in a file. Another
way of stating this is, "we do not know the uses to which the IGES Entities have been put". The
reason that the meanings are not known is that IGES is essentially a "one schema" device. It has
been developed primarily as a transform from one internal schema, a source CAD system, to
another, the IGES transfer file.

Some application committees have attempted to inject some meaning through defined properties
such as the Electrical Committe with the "Layer Property" to indicate that certain elements
represent circuitry on specific layers of a multi-layer printed circuit board. However,
without the use of a Conceptual Schema, these will remain islands of meaning without
integration.

The only way in which equal functionality could be established would be to develop a Conceptual
Schema covering the application usages and establish mandatory mappings to specific elements
of an IGES file. This would result, effectively, in converging IGES and PDES. I believe that the
result would neccessitate the production of an IGES file which would probably not meet the
concept of "upward compatibility" because it would remove options already permitted to
existing implementations. It would also demand introduction of knowledge to the pre-processor
which is not now a part of implementations (an example would be an application layering
scheme for a Computervision usage).

For example, in one enterprise performing the design of mechanical parts, there might be a
scheme that layer 10 of the CAD system file is reserved for elements which represent
construction lines used as references for the construction of the geometric model of a part. In
another enterprise, the choice might be for layer 20 to carry the same information. All that is
known in the corresponding IGES files would be that there are entities which have a level
association. There is no knowledge that the geometry on level 10 in one case, and level 20 in the
other, is not part of the definition of the shape and size of the part. In IGES today, this sort of
meaning must be transmitted from the sender of the IGES file to the receiver by memo or some
other means before the file can be correctly interpreted by the receiver.

It is true that today, different CAD systems have different devices for capturing user-defined meanings. What is designated by assignment of CAD entities to a layer in one system may be accomplished with the use of a user-defined property in another. There is reason to believe that, for PDES, there may not be a simple map from a CAD system "layer" to a "level" as there is now in IGES. When the data meanings are understood through the use of a Conceptual Schema, the "level" may not be found to be an appropriate device for such a meaning.
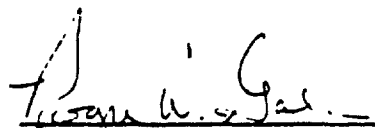
Phil Kennicott keeps insisting to me that there is a conceptual schema for IGES. I keep trying to tell him that I will believe that when I see it. By "see it" I mean expressed in IDEF1X or IA which are the only two languages I find with a comprehensible format for expression of most of the content required for a Conceptual Schema.

An IGES "Level" or "Group Associativity" is not an IDEF1X Conceptual Schema Entity or Attribute. It is an Internal Schema, physical file entity which may take on many meanings depending upon the source application, or even individual user. If the user "grouped" some geometric entities with the intent that they defined a hole, then the transfer file should tell me that this is a "hole defining group" not just a group without a reason for being grouped.

I believe that because PDES is going to be developed with the three-schema concept from the beginning and IGES was mostly one-schema and at best two-schema, there is a low probability that there will be unambiguous mappings between them. Everyone involved in PDES including the Steering Committee needs to develop a deeper understanding of the differences between IGES and PDES before making pronouncements of schedules and relationships.

I am concerned that we are about to embark on PDES with a very unrealistic set of expectations, not only among ourselves, but to an even worse degree among the non-participants who are already making noises about PDES being **the** integration technology pill

I suggest that this is a topic which ought to be on the Agenda for the next General Meeting and probably for subsequent General Meetings.

Roger W Gale
Consultant
The D Appleton Co., Inc.
1104 Highland Ave., Suite.
Manhattan Beach, CA 90266

Phone 213-318-2451